An-Najah National University

Faculty of Graduate Studies

DECISION TREE TO CLASSIFY JETS OF PARTICLES IN HIGH-ENERGY PHYSICS

By

Marwa Abdel Fattah Ahmad Khaled

Supervisor

Dr. Baker Abdulhaq

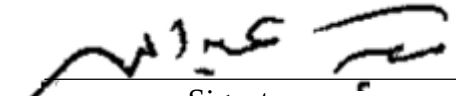
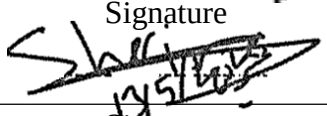
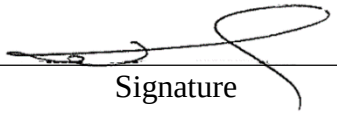
This Thesis is Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Advanced Computing, Faculty of Graduate Studies, An-Najah National University, Nablus, Palestine.

2025

DECISION TREE TO CLASSIFY JETS OF PARTICLES IN HIGH-ENERGY PHYSICS

By
Marwa Abdel Fattah Ahmad Khaled

This Thesis was Defended Successfully on 10/04/2025 and approved by

Dr. Baker Abdulhaq	
Supervisor	Signature
Dr. Shereen Hijazi	
External Examiner	Signature
Dr. Ahmad Awad	
Internal Examiner	Signature

Dedication

To my dearest parents, my beloved siblings, and my husband Saed.

Thanks for the profound impact each of you has had on my life. You are the roots that ground me and the wings that lift me up. With all my love and gratitude.

Acknowledgments

I would like to express my heartfelt gratitude to my supervisor, Dr. Baker Abdulhaq, for his invaluable guidance and support throughout this journey. His insights, encouragement, and vision were instrumental in bringing this thesis to fruition.

I am also deeply appreciative of the dedicated faculty who taught and inspired me during my master's program, with special thanks to Dr. Amjad Hawash, Dr. Ahmad Awad and Dr.Othman Othman.

Last but not least, I am grateful to my lovely family, especially dearest parents whose prayers surely had an effect in giving me the power to undertake this journey. I am also grateful to my dear husband Saed Saket and my friends Iman Mkheimer and Iman Othman for their infinite support during the writing of this thesis.

Declaration

I, the undersigned, declare that I submitted the thesis entitled:

DECISION TREE TO CLASSIFY JETS OF PARTICLES IN HIGH-ENERGY PHYSICS

I declare that the work provided in this thesis, unless otherwise referenced, is the researcher's own work, and has not been submitted elsewhere for any other degree or qualification.

Student's Name: مروة عبدالفتاح أحمد خالد

Signature: مروة خالد

Date: ٢٠/٤/٢٠٢٥م

List of Contents

Dedication iii
Acknowledgments iv
Declaration v
List of Contents vi
List of Tables viii
List of Figures ix
List of Appendices x
Abstract xi
Chapter One: Introduction and Theoretical Background 1
1.1 Problem Statements and Objectives 1
1.2 Large Hadron Collider (LHC) 2
1.3 Understanding Jets in Particle Physics 4
1.3.1 Definition of Jets 4
1.3.2 Jet Algorithms 4
1.3.3 Jet Properties and Classification 4
1.3.4 Classifying Jets into Five Nuclear Particle Categories 5
1.4 Research Questions 6
1.5 Basic Concepts of Machine Learning 6
1.5.1 Machine Learning 6
1.5.2 Supervised Learning 11
1.5.3 Classification 12
1.6 Decision Trees (DT) 16
1.6.1 Introduction to Decision Trees 16
1.6.2 Decision Tree Algorithms 17
1.6.3 Decision Trees Work 20
1.6.4 Attribute Selection Measures 21
1.6.5 Pruning in Decision Trees 23
1.6.6 Applications for Decision Trees 24
1.6.7 Advantages and Disadvantages of a Decision Trees 25

1.7 Random Forest (RF) 26
1.7.1 Introduction to Random Forest 26
1.7.2 Random Forest Work 28
1.7.3 Feature selection with Random Forests 29
1.7.4 Advantages and Disadvantages of Random Forest 30
1.8 Literature Review and Related Work 31
Chapter Two: Methodology 35
2.1 Research Design 35
2.2 The Dataset 35
2.2.1 Data Sources 35
2.2.2 Data loading and Pre-Processing 36
2.3 Decision Tree and Random Forest 38
2.3.1 Model Training 39
2.3.2 Development and Training of DT and RF Models for Jet Classification Utilizing the Scikit-learn (Sklearn) Framework 43
2.3.3 Optimization 45
2.3.4 Model Evaluation 45
2.4 Sklearn 49
2.5 Methodological Limitations 50
Chapter Three: Results and Discussion 51
3.1 The Environment of Experiment 51
3.2 Decision Tree (DT) 52
3.3 Random Forest 65
Chapter Four: Conclusion and Future Work 75
1.4 Conclusion 75
2.4 Future Work 76
List of Abbreviations 78
References 79
Appendices 84
الملخص ..... ب

List of Tables

Table 1.1: Applications of Machine Learning 8
Table 2.1: Confusion Matrix 46
Table 3.1: A Summary of the Hyperparameters Used for DT 52
Table 3.2: Accuracy of Different Criterion Functions of DT 53
Table 3.3: Summary for Two Groups (Entropy vs. Gini) DT 54
Table 3.4: Anova: Single Factor for Two Groups (Entropy vs. Gini) DT 55
Table 3.5: Accuracy of Different Splitter Functions of DT 56
Table 3.6: Accuracy of Varying Maximum Depth of DT 58
Table 3.7: Accuracy of Varying Minimum Samples Split and Maximum Depth of DT 60
Table 3.8: Accuracy of Varying Minimum Samples Leaf of DT 62

List of Figures

Figure 1.1 Classification of Machine Learning Algorithm..... 9
Figure 1.2: Decision Tree Structure..... 16
Figure 1.3: Decision Tree Construction..... 20
Figure 1.4: Entropy Function..... 21
Figure 1.5: Random Forest Simplified 27
Figure 1.6: General scheme of Random Forest 27
Figure 2.1: Flowchart of a machine learning model..... 43
Figure 3.1: Decision Tree with default parameters Code 53
Figure 3.2: Chart Accuracy of Different Criterion Functions of DT..... 54
Figure 3.3: Chart Accuracy of Different Splitter Functions of DT 56

List of Appendices

Appendix A: Figures 84
Appendix B: Tables 90

DECISION TREE TO CLASSIFY JETS OF PARTICLES IN HIGH-ENERGY PHYSICS

By
Marwa Abdel Fattah Ahmad Khaled
Supervisor
Dr. Baker Abdulhaq

Abstract

This study investigates the application of Decision Tree and Random Forest models to tackle the challenge of jet classification in particle physics, focusing on categorizing jets into five key particle types: light quarks (q), gluons (g), W and Z bosons, and top quarks. Using a dataset from Zenodo comprising list of 53 High-Level Features derived from jet events, we explored Decision Tree and Random Forest models within the machine learning framework to enhance classification accuracy. Our approach involved building and comparing various Decision Tree and Random Forest models, assessing configurations such as tree depth, the minimum number of samples leaf, and the number of trees in the Random Forest ensemble to optimize performance.

In our jet classification research, the Random Forest model outperforms the Decision Tree model in classifying particle physics events, achieving higher precision, recall, F1-scores, and an overall accuracy of (85.32%) compared to (81.32%). Optimal Random Forest performance was obtained using 100 trees, maximum depth = 10, and top 20 selected features, which also reduced training time by 36.8%. In contrast, the best Decision Tree configuration used maximum depth = 8 and maximum features = 40. This research highlights the potential of Random Forest to achieve high jet classification accuracy and offers insights into optimizing Random Forest models for similar tasks in particle physics research.

Keywords: jet classification, Decision Tree, Random Forest, particle physics, machine learning.

Chapter One

Introduction and Theoretical Background

This introduction chapter starts by explaining the main research problem, then it explores concepts related to Large Hadron Collider (LHC) and jets in an attempt to help the reader understand the study's focus on jet classification.

The chapter also explains some fundamental concepts related to machine learning (ML) and it is various approaches. Such as decision tree (DT), and random forests (RF) algorithms specifically how it is used in high-energy physics (HEP) to classify particles in the LHC. In addition to this, it will discuss the various applications of DT and RF, the author's earlier work on this subject, and the application of DT in HEP. Specifically, it will discuss the field's ongoing challenges and how our research solved these challenges in the form of a classification system for jets.

This thesis compares our work with the thesis titled "Using Artificial Neural Networks (NN) to Predict Particle Type in HEP " to highlight the complementary roles of these approaches in advancing particle physics research. While the referenced study employs NN to classify jets into the same category’s light quarks, gluons, W and Z bosons, and top quarks focusing on deep learning techniques to capture intricate patterns in particle data, our research investigates interpretable and robust alternatives through tree-based models.

1.1 Problem Statements and Objectives

High-energy physics experiments, particularly those carried out at particle colliders such as the LHC, produce intricate collision events that generate jets of particles. The precise identification and characterization of these jets are essential for advancing our understanding of fundamental particles and their interactions and it helps us understand fundamental forces such as the strong nuclear force. It is also used to reconstruct known particles such as the Higgs boson or the top quark. Conventional jet identification algorithms often struggle to capture the subtle features and complex interactions within jets, particularly in high-density and intricate environments.

This research focuses on the development and evaluation of classification models based on DT and RF algorithms to accurately classify particle jets in HEP experiments.

Particle jets are clusters of particles that emerge in the same direction following collisions in particle accelerators. For instance, when hadrons collide, quarks and gluons are produced, as observed in proton-proton collisions at the LHC at CERN, resulting in enormous datasets that require advanced methods for processing and analysis. Misidentifying jets can lead to incorrect conclusions about new particles or rare operations [40].

The proposed model aims to distinguish between different types of jets based on their distinct features. The classification framework includes five primary classes: light quarks (q), gluons (g), W bosons, Z bosons, and top quarks (t). The DT and RF algorithms are anticipated to surpass traditional methods by effectively capturing the intricate interactions within jets, thereby enhancing the accuracy of physics analyses and contributing to advancements in particle physics research. To evaluate the model's performance, key ML metrics such as accuracy, precision, recall, and F1 score will be utilized.

A comprehensive understanding of the strengths and limitations of the RF algorithm is critical for determining its suitability based on the specific characteristics of the dataset and the objectives of the ML task. This research underscores the potential of these algorithms to improve jet classification and support discoveries in HEP.

1.2 Large Hadron Collider (LHC)

The Large Hadron Collider is the world’s most powerful particle accelerator, designed to explore some of the deepest questions in particle physics. CERN existing near Geneva, Switzerland, the LHC has a circumference of approximately 27 kilometers. Hadrons, such as protons or heavy ions, are accelerated by LHC, to near the speed of light. After that, collides them at four distinct points where the machine's two beams intersect. Subatomic particles can be produced from these high-energy collisions, which can be achieved and analyzed by highly advanced equipment, among these particles there are jets, which are collimated sprays of particles resulting from the fragmentation of quarks and gluons produced by high-energy collisions.

Fundamental physics can be understood significantly by LHC, it is also enabled the finding of the Higgs boson and give insights into the forces that govern particle interactions. Energy information’s are collected from these experiments, momentum,

and angles of the produced particles, is vast and complex. To advancing understand the universe at very high and fundamental level, it is crucial analyzing this data. So that to classify and interpret of the particle jets which resultant from these collisions. This can be happened by using ML models such as DT and RF. In these case scientists who work in this field can classify jets more accurately, they can achieve meaningful pattern for these data, which leading for particle physics with more precise models, and it may be led to discover new physics phenomena [30].

- **Design and Construction of the LHC**

In the early of 80ths from previous century, he Large Electron-Positron collider was initially imagined and constructed, The CERN research teams put a powerful plan for future, and it is Council agreed the construction of the LHC in December 1994, achieving CERN's longstanding aspiration for a robust collider.

The approval of this project came from its fixed budget. Whereas enhancement and improvement in this project came from contributions from non-member states. Initial budget constraints suggested a two-stage construction process for the LHC. However, following additional support from Japan, the USA, India, and other countries, the CERN Council approved a single-phase construction plan in 1995. Between 1996 and 1998, four primary experiments ALICE, ATLAS, CMS, and LHCb received official approval, allowing construction to proceed on the LHC's four main sites. Later, three smaller experiments TOTEM, LHCf, and MoEDAL were added, each positioned close to one of the main detectors [30].

Collision Energy at the LHC: Why is it Significant? The LHC was designed for a maximum proton beam energy of 7 TeV, producing a collision energy of up to 14 TeV between two protons. For lead-ion beams, the collision energy reaches an impressive 1150 TeV due to the large number of protons in each lead ion. These collision energies, while not exceptionally high in absolute terms (1 TeV is roughly equivalent to the energy of a flying mosquito), are unique because they are concentrated into a microscopic space millions of times smaller than that of a mosquito. This high energy concentration is what makes the LHC's particle collisions particularly valuable for scientific exploration [30].

1.3 Understanding Jets in Particle Physics

1.3.1 Definition of Jets

In HEP, jets refer to tightly collimated streams of particles generated during particle collisions, such as those at the LHC. When high-energy particles collide, they produce secondary particles that generally move in the same direction, forming jets. Analyzing these jets provides insights into particle interactions and energy/momentum exchanges, deepening our understanding of fundamental physics [35].

1.3.2 Jet Algorithms

Various algorithms are used to analyze jets in particle physics. These algorithms classify and group particles produced in high-energy collisions. Notable jet clustering algorithms include:

- kT Algorithm: Groups particles by their proximity in momentum, focusing on high-momentum particles.
- Cambridge/Aachen Algorithm: Clusters particles based on spatial distance, using angle and separation.
- Anti-kT Algorithm: Forms circular-shaped jets in rapidity-azimuth space, making it practical for detailed jet analysis.
- SISCone Algorithm: Clusters particles within cone-shaped areas, useful for high-density particle environments.

These algorithms are crucial for identifying jets in experimental data and advancing particle physics knowledge [27].

1.3.3 Jet Properties and Classification

Jets are classified based on various properties that help distinguish them during particle collision analysis. In total, 53 key properties are used to "tag" jets, aiding in identifying particle types and characteristics. These properties include:

1. Basic Kinematic Features: Fraction of the jet's transverse momentum, Transverse momentum of the jet, Pseudorapidity of the jet and Invariant mass of the jet.
2. N-subjettiness Variables: N-subjettiness variables with beta=1, N-subjettiness variables with beta=2, Ratio of tau3 to tau2 for beta=1 and Ratio of tau3 to tau2 for beta=2.

3. Energy Correlation Functions (ECFs): ECFs for beta=0, 1, 2, Second-order ECFs for beta=1, 2, Ratios of ECFs for beta=1, 2, Angular variables combined with ECFs, Multiplicative combinations of ECFs and Normalized combinations of ECFs.
4. Modified Mass Drop Tagger (mMDT) Variables: N-subjettiness variables with mMDT for beta=1, N-subjettiness variables with mMDT for beta=2, Ratios of tau3 to tau2 with mMDT for beta=1, 2, ECFs with mMDT for beta=0, 1, 2, Second-order ECFs with mMDT for beta=1, 2, Ratios of ECFs with mMDT for beta=1, 2, Angular variables combined with ECFs with mMDT, Multiplicative combinations of ECFs with mMDT, Normalized combinations of ECFs with mMDT.
5. Mass Variables: Various jet mass calculations using different grooming techniques.
6. Other Features: Number of constituents in the jet.

Data on these properties, available on platforms like Zenodo, is used to train DT and RF that classify jets according to these properties.

1.3.4 Classifying Jets into Five Nuclear Particle Categories

The objective of this project is to classify jets into five categories of nuclear particles light quarks (q), gluons (g), W and Z bosons, and top quarks using DT and RF trained for high accuracy. Here is a brief description of each category:

1. Light Quarks (q):
- Fundamental particles in protons and neutrons, including "up" and "down" quarks.
- Interact with both weak and strong nuclear forces, playing a key role in nuclear structure.
2. Gluons (g):
- Particles that carry the strong nuclear force, binding quarks within atomic nuclei.
- Interact through complex processes governed by quantum chromodynamics.
3. W and Z Bosons:
- Act as carriers for the weak nuclear force, enabling particle transitions within nuclei.
- Play a role in nuclear decay and other particle transformation reactions.
4. Top Quarks:
- Heaviest quarks, interacting with both weak and strong nuclear forces.
- Mostly produced in high-energy collisions, such as those occurring in nuclear experiments.

These classifications enhance our understanding of particle physics, and DT and RF-based categorization provides a robust approach to identifying particle types and interactions in collider experiments.

1.4 Research Questions

The research aims to answer the following research questions:

1. Is DT a reliable method for conducting a jet classification?
2. What are RF parameter settings yielding the highest accuracy for jet classification?
3. What is the most efficient way of using DT or RF for carrying out jet classification?
4. What is the minimum set of features needed to be used?
5. What are the most significant features for jet classification?
6. What is the best way to construct (i.e., train) DT or RF to classify jets?

1.5 Basic Concepts of Machine Learning

This section provides an introduction to the concepts that underpin ML as well as an overview of the various methods of learning, including supervised, unsupervised, semi-supervised, and reinforcement learning. In addition to this, it will discuss the classification problem that arises in ML and how the DT and RF algorithms can be utilized to find a solution to this problem.

1.5.1 Machine Learning

In 1959, Arthur Samuel provided a definition of ML by describing it as the process of putting data into computer systems in such a way that the computer learns to understand and perform the task without being explicitly programmed or given similar data [52].

"Predictive analytics" or "statistical learning" are two names that have been given to the subfield of research that brings together elements of computer science, artificial intelligence, and statistics. In the most recent few years, methods of ML have steadily made their way into more and more aspects of day-to-day life. Many contemporary websites and devices rely heavily on ML algorithms to function properly. For instance, websites are able to make automatic recommendations for movies to watch, food to order, or products to purchase. Personalization is possible with online radio, and your friends can be located through your photo album [10].

ML is the study of, and development of, mathematical models and algorithms that can learn from data and improve based on what they already know to get the best results [48]. This field of study and development is known as "machine learning."

Because models cannot change themselves in response to new data, iterative learning in ML is an extremely important component of the field. They are able to give accurate results because they learn from the data they have already collected and because they can repeat their results. Although it has been studied for a long time, there has been a lot of interest in it in recent years [48].

The field of ML is one that is experiencing rapid expansion. Without even realizing it, we make use of ML on a daily basis in applications such as Google Maps, Google Assistant, Alexa, and many others. Table 1.1 outlines some of the more well-known applications of ML in the real world, including [5].

Table 1. 1

Applications of Machine Learning

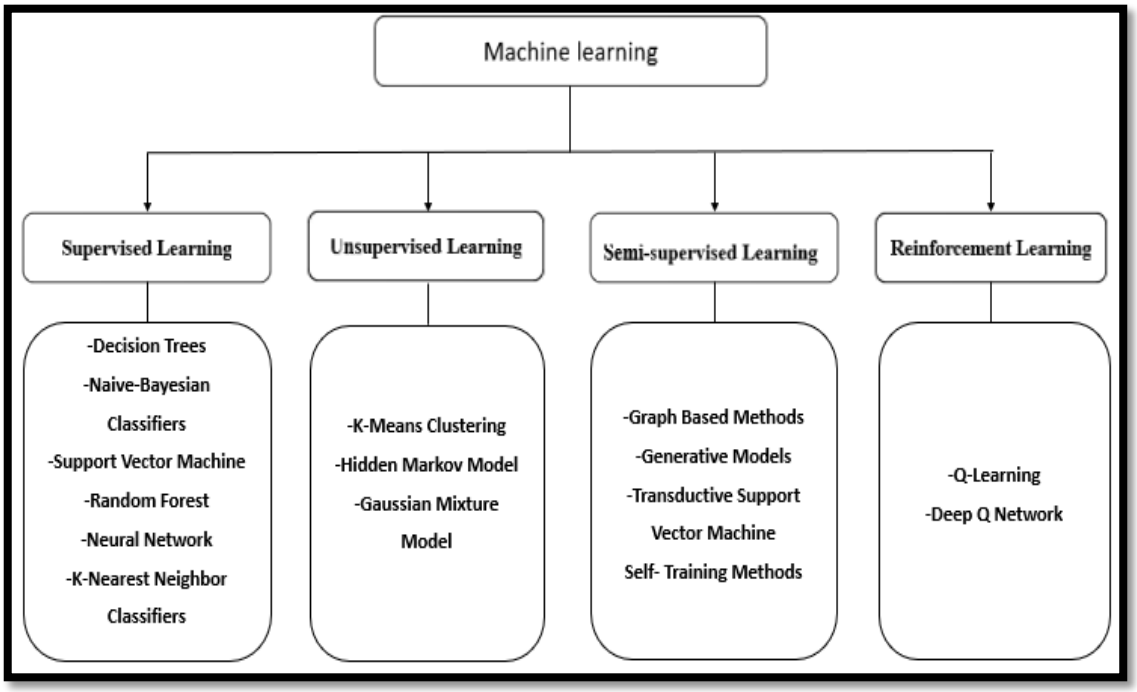
Application	Description
Playing Checkers Game	A computer program acquires the skill to play checkers by continually enhancing its performance, gauged through its success in diverse tasks related to the game. This improvement is achieved through self-play, allowing the program to accumulate experience and refine its strategies.
Speech Recognition	Contemporary speech recognition systems, such as the SPHINX system, leverage ML algorithms. For instance, SPHINX learns using NN methdologys to learn voices of speaker-specific and words from speech signals. to interpret hidden Markov models. This customization applies to adapting to individual speakers, improving dictionaries, and managing background confusion.
Autonomous Vehicles	Many systems and autonomous vehicles such as cars and drones operated by ML models, this applies to Google's Driverless Cars and Tesla Cars. It also shows its effectiveness in sensor-based applications.
Filtering Emails (Spam Emails)	Filter spamming one of the most important ML application by memorizing patterns from previous spam emails identification. So that when new spam come to inbox the model use machine learning pattern to classify it as an spam email.
Robotics and Artificial Intelligence	ML consider as very useful approach for problem solving by building knowledge from training data, then improve leaning model to achieve very high level robotics and artificial intelligence.
Web and social media	• Text mining as spam filtering or categorizing web pages, emails, or documents are solved by Naive Bayes classifier it is very useful and successful. • Naive Bayes classifier are used in Facebook application to analyze and recognize expressions of positive and negative emotions. • Document categorization in Google application use Naive Bayes to classify documents. • K-means clustering as a ML model are used to group web pages depending on similarity in search engines as Google and Yahoo. • Items that are frequently purchased together in e-Comers websites such as Amazon or Flipkart used recommendation system based on Apriori which are built by ML models. • Google's auto-complete feature also leverages Apriori which are built by ML models. As a user types a word, the search engine looks for associated words that commonly accompany the previously typed word. • Sentiment analysis on social networking sites addresses text classification challenges, employing a variety of ML algorithms.
Medical Field	In the medical field, the Trauma & Injury Severity Score (TRISS) predicts mortality in injured patients and was initially developed using logistic regression. Logistic regression is also used in creating various medical scales to assess patient severity.
Bayesian Methods	Bayes' Theorem is a popular method for calculating conditional probabilities, often integrated with various ML models and algorithms to solve complex data science and analytics problems. Real-world applications of Bayesian methods include Netflix suggestions, auto-correction, credit card fraud detection, Google page ranking, facial recognition, climate modeling forecasts, and stock trading systems.
Physics Field	ML and it is various approaches. Such as DT algorithm, and RF algorithm can be used in HEP to classify particles in the LHC, as we will discuss in this thesis.

As was just mentioned, the field of ML is a vast and intricate science that can be segmented off into a variety of subfields and subspecialties. At first glance, it can be partitioned into four distinct learning processes, which are as follows [52]:

1. Supervised Learning.
2. Unsupervised Learning.
3. Semi-supervised Learning.
4. Reinforcement Learning.

Figure 1.1

Classification of Machine Learning Algorithm [2]



1. Supervised Learning:

In supervised learning data scientists provide input, output and feedback to build model. This research concern in supervised learning models which can be achieved using classification. It will have discussed briefly in section 1.4.2.

2. Unsupervised Learning:

In this type of learning, Data (X) is used as input; however, there are no labels involved. Learning is the process of trying to figure out what the data mean by grouping together samples that are similar to one another or making assumptions about correlations.

Unsupervised learning is a way of referring to the process of learning in an algorithm where there is no instructor. Unsupervised learning can take two forms: association learning and clustering learning. The clustering analysis reveals how the data are organized. On the other hand, association illustrates the principles that underlie the connections between the events [7].

Unsupervised learning is a great way to draw new conclusions from large data sets and provide supervised machine learning algorithms with new data to work with. This is because supervised machine learning algorithms are designed to learn from examples. A few examples of algorithms for learning without being directly observed are as follows:

K-means clustering algorithm, Gaussian Mixture Model, and Hidden Markov model [22].

3. Semi-supervised Learning:

This method of education falls somewhere in the middle of being observed and being left to one's own devices [7]. Both labeled and unlabeled data are presented in a predetermined order for the purpose of semi-supervised learning. The amount of data that has been labeled is relatively low, but there is a significant amount of data that has not been labeled. The data are utilized to construct an accurate model for the purpose of data classification. The unlabeled data are organized into groups in order to achieve the goal of semi-supervised learning, which is to sort the data based on the labeled data. Quite a few of the most well-known algorithms are utilized in the process of semi-supervised learning e.g. clustering algorithms, graph-based algorithms and generative models. Methods that involve self-training, methods based on graphs, and generative models [22].

4. Reinforcement Learning:

In the same way that unsupervised learning uses input and either positive or negative feedback, reinforcement learning does the same thing. How the algorithm functions determines whether the feedback is positive or negative. When the algorithm needs to make decisions on its own based on the results of its actions, trying to get good results and avoid making mistakes, this type of learning, which is basically the same as trial

and error, can be useful [17]. Common examples of ML algorithms that use reinforcement include Q-learning, SARSA, and Deep Q-Networks.

1.5.2 Supervised Learning

"Supervised learning" is a term that's frequently used in the field of ML to refer to methods for developing predictive models [33]. Modeling the interactions that occur between inputs and outputs is the primary objective of supervised learning, which is a subset of automated learning [21]. When an algorithm is using supervised learning, it has a target and an outcome variable that must be predicted from a set of predictors that serve as independent variables. This can be accomplished by using a variety of independent variables to make predictions. You will be able to create a function with these sets of variables by using it to transform the inputs into the outputs that you require. This training procedure is carried out several times until the machine-learning model achieves the level of precision that is required for it to function properly on the training data [17].

The supervised procedures have applications in a variety of fields, including manufacturing, banking, marketing, medical, and physics, amongst others. Classification models (classifiers), on the one hand, and regression models, on the other, are the two primary types of supervised models that should be differentiated from one another. Input space is converted into a set of real numbers by regression models, which take this space as their starting point. A regressor, for instance, can estimate the amount of demand that will be placed on a product by consumers by considering the benefits that the product possesses. Classifiers are used to assign the input space to categories that have previously been established. Classifiers can be utilized, for instance, to differentiate between good mortgage borrowers (those who pay back their loans on time) and bad mortgage borrowers (those who do not pay back their loans on time) [33]. In medical field Classifiers can be utilized to differentiate between cancer cells and non-cancerous cells. Other example, in physics field can be assigned to classify particle jets in high energy physics experiments. Jets are collections of particles that are propelled in the same direction and are produced by particle accelerators**.** The following is a list of some examples of these different learning algorithms: DT, Rule-Based Classifiers, Naive-Bayesian Classifier, k-Nearest Neighbor Classifiers, NNs, Linear Discriminant

Analysis, and Support Vector Machines are some of the classification methods that have been developed. SVMs [22].

1.5.3 Classification

Classification is a technique used to predict similar information based on the values of a categorical target or class variable. It is a valuable method for analyzing various types of statistical data. These algorithms have diverse applications, including image classification, predictive modeling, and data mining [44][2]. It is possible to guess discrete responses by using classification tasks. It is beneficial to have the ability to categorize, tag, or otherwise organize the data into a variety of different classes or categories.

Classification is utilized in a variety of fields, including medical imaging, speech recognition, and bank credit scoring, to name a few of the more prominent or prevalent examples. Additionally, categorization is utilized in the process of deciphering handwritten letters and numbers, determining whether or not an email is legitimate or spam, and even diagnosing whether or not a tumor is cancerous [43].

Classification is a controlled machine learning method used to predict input data marks. It uses educational data and test data to study properties and predict class in unknown data sets. The algorithm involves two stages. [53]. The steps listed below can be used to create a classification model [45]:

1. Clearly describe the Problem: define the classification problem and the classes/categories of interest.
2. Gathering and Preparing Data: As you gather information for the classification process, make sure it is accurate and comprehensive. As necessary, deal with outliers, missing numbers, and inconsistent data.
3. The dataset are spitted in to training, validation, and testing data. The validation set is used for hyperparameter adjustment (if applicable), the test set is used for the final model evaluation, and the training set is used for training the model.
4. Classification technique must be selected based on problem specifications, the features of the dataset, and the software and hardware resources. the common ML classification models NNs, Support Vector Machines, RF, DT, and Logistic Regression.

5. Any ML approach need training before it generalized. So, the training datasets are used to train the selected model.
6. Grid search, random search, or Bayesian optimization are used to fine-tune hyperparameter, this operation will increase model performance.
7. To evaluate the trained model the testing, and validating datasets are used. Whereas to evaluate the performance of the model, use suitable assessment measures such ROC-AUC curve, F1-score, accuracy, precision, recall, and confusion matrix analysis.

There are essentially two categories into which classification algorithms can be subdivided, linear classification models such as logistic regression, and support vector machines, non-linear models such as k-nearest neighbors, naïve bayes, DT, and RF [53].

- **Linear Models**

Linear models are a class of statistical and ML models that assume a linear relationship between the input features and the output. In essence, these models express the relationship as a linear combination of the input features, often represented by a weighted sum, with an additional bias term. Common types of linear models include:

- **Logistic Regression**

For binary classification tasks, statistical ML models such as logistic regression are employed. LR is an algorithm for supervised learning that uses one or more input independent variables (features) to predict the likelihood of a binary result. Using the logistic sigmoid function to convert a linear combination of input data into a probability value, the basic principle of logistic regression is to model the relationship between the input features and the binary result. This transformation is appropriate for interpreting the likelihood of the positive class, which usually represents the event or outcome of interest in binary classification problems, as it guarantees that the predicted probabilities lie within the range of 0 to 1. While the training step are performed, every input feature in the model is discovered a set of coefficients, sometimes referred to as weights. The goal of optimizing these coefficients is to reduce the discrepancy between the training dataset's actual binary labels and predicted probabilities. Because it is easy to use, useful for binary classification problems, and can be applied to a wide range of fields,

including finance, healthcare, and sports analytics, logistic regression is a popular software [16].

- **Support Vector Machines (SVMs)**

SVMs are a directed learning method for classification and relapse, partitioning datasets into classes using edge calculation. They display each highlight as a point in n-dimensional space, extending the gap between adjacent information centers and the edge hyperplane [5].

Non-linear Models

Non-linear models are a class of statistical and ML models that capture complex relationships between input features and the output in a non-linear fashion. Unlike linear models, which assume a linear relationship, non-linear models can represent more intricate patterns and structures in the data. These models are essential when the underlying relationship between variables is not well-described by a straight line. Some common types of non-linear models include:

- **K-Nearest Neighbors**

This approach to regression and classification does not make use of any parameters in its calculations. The KNN algorithm uses N training vectors to locate the k-nearest neighbors of an unknown feature vector whose class needs to be determined [5].

- **Naïve Bayes**

Any time Bayes' theorem is utilized to calculate the probability of a label given some input attribute values, it is the basis for naive Bayes classification-method variations. Using Bayes' theorem and the assumption that the predictors are independent of each other, this method of classification finds the way to distinguish between things. In the realm of text classification, and really in many areas, the naive Bayes classification is one most often used. Quite often, it is used to predict the conditional probability of occurrence of an event for that occurrence and, therefore, classify accordingly. [32].

- **Decision Tree Classification**

DT are a type of predictive model that can figure out a target value based on the input data by linking data elements together in a model and making connections between

them. It is structured similarly to a typical tree in that it possesses a root node, branches, and leaf nodes [56]. DT are designed to function the same to the way people think, which enables them to consider the evidence and arrive at sound judgments [41]. In the following section, we'll go over it in even greater depth 1.5.

- **Random Forest Classification**

It is a method of ensemble learning that can be utilized for both classification and regression. The bagging method generates a great number of distinct DT, each of which is based on a unique collection of data. When creating the final DT, the output of each separate choice tree in the RF is summed up and stated together.

The RF Algorithm is comprised of two stages: the first stage is the creation of the RF, and the second stage is the application of the classifier that was developed in the first stage to the task of making a prediction [5]. We'll cover it in more depth in the section 1.6.

Different tasks fall within the category of classification:

- A classification is said to be binary when there are only two possible outcomes. For instance, recognizing spam or fraudulent activity (like determining whether an email is spam or not) and forecasting the weather (like determining whether or not it will rain).
- A problem referred to as multi-label classification is one in which there are more than two potential solutions. An illustration of this would be the fact that a single piece of news could simultaneously be classified as a sports story, a story about a particular player, and a story about a particular location [45].

Utilizing a prediction model that is referred to as a "decision tree" is one of the most effective ways to resolve classification issues. This well-known method of arranging things can be illustrated using a graph. This final quality is particularly crucial with regard to the algorithms that are used to assist in the resolution of issues. The creation of a classic DT typically involves the use of various statistical techniques. On the other hand, DT are presented in the form of graphs, and there are so many different ways to approach a problem that other methods can also be utilized [28].

1.6 Decision Trees (DT)

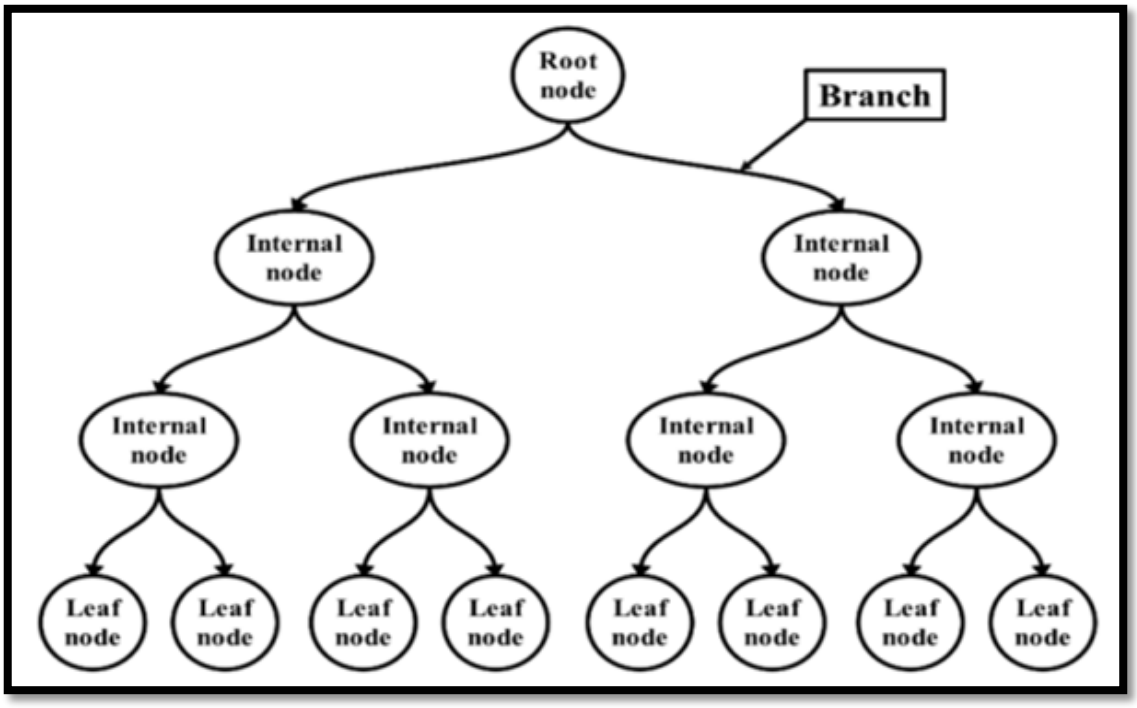
1.6.1 Introduction to Decision Trees

DT are a type of non-parametric supervised learning that can be used for both classification and regression analysis. DT are trained on labeled data to correctly classify data didn't know before. DT have some advantages over other supervised learning methods [15], which is one reason why data scientists like them. Using ML orders that were trained with training data, the main goal was to figure out the type or value of target variables (Vlahakis et al., 2020; Sarker, 2021). The method can be used without having to know a lot about statistics or complicated math formulas, and it is easy to understand even if you only know a little bit about math [29].

The DT is a technique for classifying data sets that is used to make a model that looks like a tree from the top down. This model is based on the characteristics of the data set. This model is based on what the data set says about itself. A DT is also a predictive modeling technique that uses a training set of data with the same features (attributes) to predict, classify, or categorize new data objects. For this technique to work, it needs a set of training data [23].

Figure 1.2

Decision Tree Structure



Roots, branches, and leaves are all parts of a normal tree. The format is the same for the DT. It has a root node, leaf nodes, and branches. Every internal node has a test for an attribute. The result of the test is on the branch, and the class label that comes out of the test is on the leaf node. As its name suggests, a root node is the node at the top of a tree. It is the parent of all other nodes. A DT is a tree where each node is an attribute, each branch is a rule, and each leaf is the result (a category or a value that stays the same). Because DT mimic the way people think, it's easy to gather the data and come up with some good ideas. The goal is to build an information tree like this one and process a single result at each leaf [41].

The DT classification method is done in two steps: building the tree and cutting it down. In stage one, the tree is built from the top down. At this point, the tree is split up in a way that goes back on itself until all of the data elements have the same class label. Since the training data set is always being reprocessed, this stage takes a lot of time and involves a lot of computations. The bottom-up method is used to prune a tree. This procedure improves the accuracy of the algorithm's predictions and classifications by reducing overfitting [23]. One of the key challenges in training a ML model is preventing overfitting, which, if it happens, will lead to low accuracy rates. The model is overly attempting to identify any noise in the training dataset, which is the cause of this problem. The data points that inaccurately depict the characteristics of the data are referred to as noise. As a result, a highly specialized model is produced that works well on training data but struggles to generalize to new data [8].

1.6.2 Decision Tree Algorithms

There are several algorithms, like CHID, CART, ID3, C4.5, and others, that are used to build DT. We'll talk about a few common algorithms and what's good and bad about each one.

- **ID3**

Quinlan (1986) says that the ID3 method is a very easy DT algorithm. The ID3 algorithm stops growing when all instances belong to a single value of a target feature or when the best information gain is less than zero when using information gain as a splitting condition. ID3 doesn't deal with numeric attributes or missing data, and it doesn't use pruning techniques either.

ID3's best feature is that it is easy to use. Because of this, the ID3 algorithm is often used in educational settings. ID3 does have a few problems, though:

1. Since ID3 uses a greedy method, it doesn't always find the best solution and can get stuck in local optimums. Backtracking could be used during the search to stay away from the local best solution.
2. ID3 can get too good at fitting the training data. To avoid overfitting, smaller DT should be used instead of bigger ones. This method of Data Mining with DT often leads to short trees, but not always.
3. ID3 was made with the idea of nominal attributes. So, if you want to use continuous data, you must first turn it into nominal bins.

Due to these problems, most professionals choose the C4.5 algorithm over ID3, which is an improvement on ID3 that tries to fix its problems [33].

In C4.5, which is an improvement on ID3 and was made by the same author [Quinlan, 1993], the splitting rule is the gain ratio. When less than a certain number of instances need to be split, splitting stops. After the growth phase, pruning based on mistakes is done. C4.5 can work with attributes that are numbers.

In many ways, the C4.5 algorithm makes ID3 better. These are the most important changes:

1. C4.5 uses a pruning method in which branches that don't help accuracy are cut off and replaced with leaf nodes.
2. Attribute values can be missing if C4.5 is used.
3. To handle continuous attributes, C4.5 splits their value range into two subsets (binary split). The best threshold is the one that meets the gain ratio requirement with the most gain. All values above the threshold are in the first subset, and all other values are in the second subset.

It is said that C5.0 uses memory and calculation time much more efficiently than C4.5. C5.0 is a commercial version of C4.5 that has been changed and improved in a number of ways. In some cases, it can speed up the process from an hour and a half (as it did with the C4.5 algorithm) to just 3.5 seconds, which is a huge improvement.

The open source J48 library is used to make the C4.5 method used by the Weka data mining tool work in Java. Since J48 is just a reimplementation of C4.5, it is expected that it will work about the same as C4.5. But recent research that compares C4.5, J48, and C5.0 (Moore et al., 2009) shows that C4.5 is usually more accurate than C5.0 and J48, especially on small datasets [33].

- **CART**

CART, which stands for Classification and Regression Trees, was developed by Breiman and his colleagues in 1984. The fact that it is able to construct binary trees, in which each internal node is connected to exactly two edges going out, sets it apart from other algorithms. The splits are selected according to two different criteria, and the cost-complexity method is utilized in the process of pruning the resulting tree.

When the input is provided, CART has the capability of taking into account the costs of misclassification when it triggers the tree. Users also have the option of providing a probability distribution based on historical data.

The capability of CART to generate regression trees is one of its most notable features. A regression tree is a type of tree that only has one set of leaves and is used to predict a single number rather than a group of variables. CART searches for splits in regression that will result in a smaller prediction squared error, also known as the least–squared deviation. The forecast for each leaf is derived from the weighted mean of each node which serves as the basis for the prediction [33].

- **CHAID**

CHAID is one central algorithm for DT learning. It was developed in 1980 by Gordon V. Kass. CHAID differentiates and identifies interactions among variables while being an easy method to learn. CHID stands for the extension of AID and TIDE techniques. [33].

CHAID treats missing values as a single acceptable category because it was initially solely intended to handle notional attributes. CHAID doesn't carry out any pruning [35].

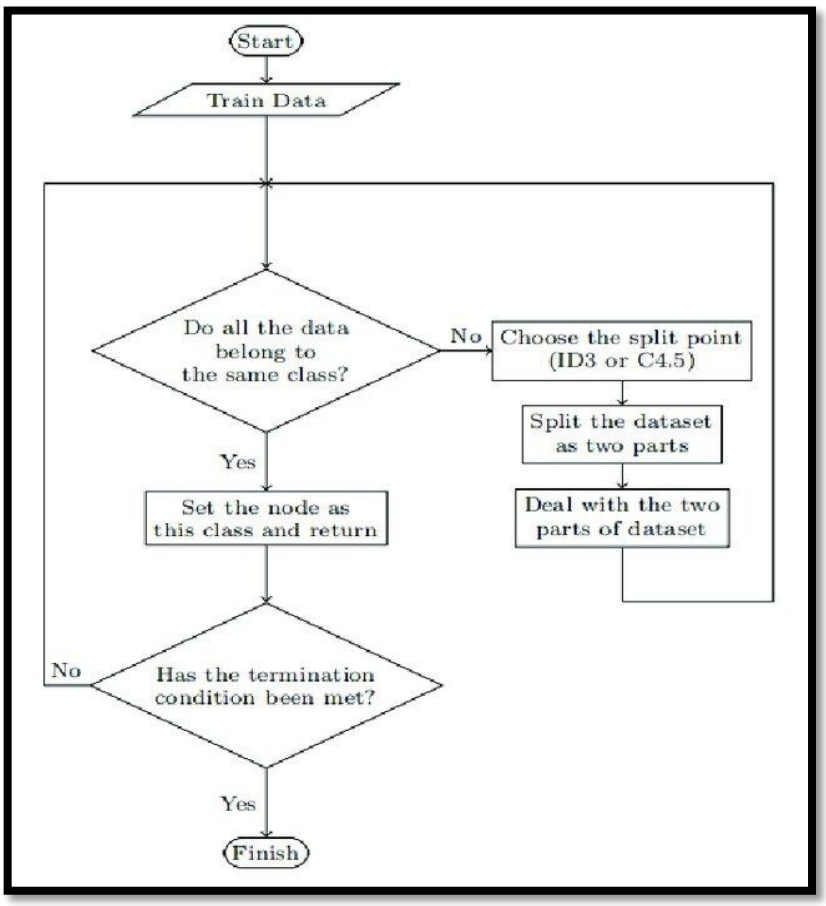
1.6.3 Decision Trees Work

A DT is constructed utilizing a top-down recursive method as well as a divide-and-conquer strategy, as shown in Fig. 1.3. The root node of the tree represents the entirety of the training dataset and it is where the tree begins [3].

1. If the results of the training lists are the same, the node will be considered a leaf, and it will be assigned the class corresponding to that status.
2. If this is not the case, the tree divides the set according to the attribute that contains the most information and assigns a name to each node.
3. Perform the procedure as many times as necessary until all of the samples belong to the same class, there are no more samples, or there are new attributes for the section.
4. Tree Ends.

Figure 1.3

Decision Tree Construction [17]



1.6.4 Attribute Selection Measures

Determining the optimal placement of attributes within a dataset, specifically selecting the attribute to be placed at the root of a tree and assigning others to internal nodes at different levels, can pose challenges when the dataset contains N attributes. The problem cannot be resolved solely by assigning a single node as the root. If the method lacks precision, the outcomes may prove to be unsatisfactory.

The researchers have successfully devised several innovative methodologies to address the issue of attribute selection. It was suggested to employ criteria such as:

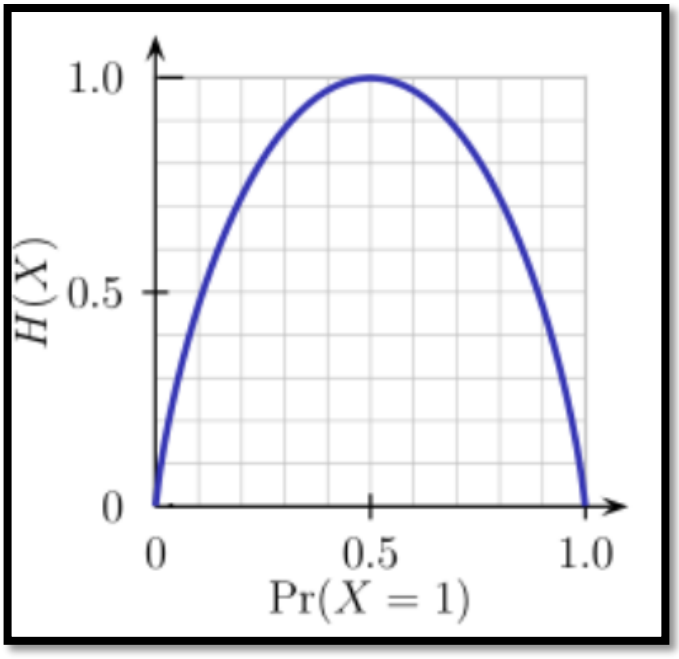
- **Entropy**

It is a measurement from the information theory to determine how randomly dispersed an attribute's possible values are. The attribute's entropy increases with the degree of unpredictability. The highest probability of an event is called entropy. An event lacks randomness if we are aware of its result in advance. Entropy is hence zero [3].

Figure 1.4 show entropy function. Entropy values range from 0 to 1 at all times. In other words, its value is better when it equals 0, and it is worse when it equals 1, i.e. the closer it is to 0, the better [21].

Figure 1.4

Entropy Function



The following formula is used to determine the sample data set S initial information entropy:

$$Entropy(S) = -\sum_{i=1}^{m} p_i \log_2 p_i \quad \text{..... (1)}$$

Where p_i represents the proportion of total samples that come from a particular class, and m is the total number of classes. In a scenario in which all of the data have the same class label, $m = 1$, and $p_i = 1$, the entropy is equal to 0. When each item of data has its own class label, the entropy value is equal to 1/m, which is equal to 1.

$$Entropy(S) = log_2 m \quad \text{..... (2)}$$

The sample data set S's split entropy Assume that S is divided into S_L and S_R by an attribute. The weighted entropy of each subgroup is the split entropy, given as:

$$Entropy_A(S) = \frac{|S_L|}{|S|}\text{Entropy } (S_L) + \frac{|S_R|}{|S|}\text{Entropy } (S_R)$$

Where A is any an attribute of C, S_L and $S_R$are subsets of set S split by A. $|S|$ is the number of samples in S. $|S_L|$ and $|S_R|$ are the number of samples in S_L and S_R respectively (30).

- **Information gain**

Information gain, also referred to as mutual information, is a metric utilized for segmentation. This statement provides an intuitive explanation of the extent of knowledge regarding the value of a random variable. A higher value is indicative of superior quality, serving as the opposite of entropy [21].

The following definition captures the knowledge gain of attribute A [57]:

$$Gain(A) = Entropy(S) - Entropy_A(S) \text{..... (3)}$$

- **Gini index**

This index measures a feature's capacity for class identification. The training records' noises in D are estimated using the Gini index as follows:

$$Gin(D) = 1 - \sum_{i=1}^{m} p_i^2 \quad \text{..... (4)}$$

The probability of a tuple (in dataset D) belonging to class C_i is denoted as p_i. The selection process involves calculating the Gini Index for each feature individually and then choosing the top k features with the smallest Gini Index [26].

- **Gain Ratio**

The C4.5 method incorporates a split information value in order to mitigate overfitting and ensure equitable information gain. The user's text can be rewritten in an academic manner as follows:

$$SplitInfo\ (A) = -\sum_{i=1}^{k} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|} \ldots\ldots\ldots\ldots\ldots\ldots\ldots (5)$$

The utilization of gain ratio addresses the concern of information gain by taking into account the number of branches that will be generated prior to splitting. By taking into consideration the inherent information of a split, it effectively adjusts the measure of information gain. The representation of the information gain ratio for attribute A is denoted as [57]:

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo(A)} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (6)$$

1.6.5 Pruning in Decision Trees

Making the trees and pruning the trees are often the two key processes in modeling the DT (Bui et al., 2012). Most of the time, pruning is necessary to prevent the tree from growing in the wrong places. The algorithm that built the greatest tree and self-prunes it after determining the appropriate pruning threshold is an excellent one.

Pre-pruning and post-pruning are distinct methodologies employed for the purpose of tree pruning. In this approach, the pre-pruning technique, known as the Early Stopping Rule, is employed to halt the tree-growing algorithm prior to achieving a fully grown tree that perfectly matches the complete training set. In order to accomplish this objective, it is imperative to implement a more rigorous termination criterion. One potential approach is to cease the expansion of a leaf node when the observed increase in impurity measure (or enhancement in the predicted generalization error) falls below a designated threshold. One advantage of employing this approach is its ability to mitigate the formation of overly intricate subtrees that excessively conform to the training

dataset. However, determining the optimal threshold for early termination can present a considerable challenge. If the threshold is set at an excessively high level, it will result in models that are poorly fitted. Conversely, if the threshold is set too low, it may not adequately address the issue of model overfitting.

Furthermore, post-pruning the DT is originally developed to its greatest size in this method. The next process is tree pruning, which involves trimming the fully developed tree from the bottom up. A subtree can be trimmed by either (1) adding a new leaf node whose class label is derived from the majority class of records associated with the subtree, or (2) the most frequently used branch of the subtree. When no more progress is seen, the tree-pruning process is finished. Because post-pruning makes pruning decisions based on a fully developed tree, as opposed to pre-pruning, which may suffer from early termination of the tree-growing process, it tends to produce better results than pre-pruning. However, when the subtree is trimmed in post-pruning, the additional calculations required to construct the complete tree may be wasted [46].

1.6.6 Applications for Decision Trees

DTs have been successfully applied in a variety of fields, including healthcare, finance, and education. DT can be used to mine educational data to find hidden patterns, such as identifying inappropriate student conduct, forecasting student performance and learning outcomes, and assisting teachers in course preparation. By creating a DT from the typical customer behavior and using it to forecast fraudulent financial behavior, it can also be used in business for fraud detection.

Market segmentation to identify customers who are likely to purchase specific items and customer churn to identify customers who are likely to switch to a rival are other applications of DTs in business. Healthcare for medical diagnosis, credit card analysis, and other uses of DTs are also possible [47].

DTs and ensemble methods like RF assist in particle physics experiments (e.g., Large Hadron Collider) by classifying events, such as identifying different particles based on collision signatures and classifying galaxies based on their characteristics (shapes, sizes, spectra), contributing to large-scale surveys and astronomical data analysis.

In HEP, DT are pivotal in several applications, especially in particle identification and classification. They are widely utilized for event selection, classifying events detected in particle accelerators to effectively differentiate between signal events (events of interest) and background events (irrelevant data). DT are very useful in particle tracking, it recognize patterns that compatible to different types of particles as they move through detectors. In the other hand, they are essential for detecting outliers, identifying rare events or anomalies in large datasets that may refer to new physics phenomena. Also feature selection in DT are very easy, it can detect the most relevant features, which have an important effect in classification or regression tasks. This is particularly important in HEP, where datasets often contain a large number of features [1].

These applications highlight the significance of DTs in advancing scientific research, improving data analysis capabilities, and contributing to decision-making processes across diverse scientific fields [12].

1.6.7 Advantages and Disadvantages of a Decision Trees

DT have more attraction to data scientists because it has a number of advantages over conventional supervised learning techniques. Some advantages include there:

- High interpretability for humans (Huysmans et al. 2011; Letham et al. 2013).
- Design that is non-parametric (Murphy 2012). Regarding the underlying data distribution, these models don't make any major assumptions. They don't have set functional shapes; therefore, they are more flexible and can capture intricate patterns in the data [25].
- Relatively low cost of computing (Han et al. 2006).
- Knowledge of non-linear correlations between the attributes (Han et al. 2006). A DT is a non-linear ML model that capable of capturing intricate linkages and interactions between the many attributes in a dataset. DT work well in situations where there is non-linear separability of features, as opposed to linear models, because they can manage non-linear correlations [25].
- Adaptability to missing values (Quinlan 1993).
- The capacity to manage data that is both continuous and discrete (Quinlan 1996).
- Proficiency with non-binary labels (Quinlan 1996) [15].

- A DT requires little time and effort and performs at a high level [23].
- DT can operate on a variety of platforms in data mining applications [23].

DT, on the other hand, contain disadvantages like:

- The target attribute must only contain discrete values for the majority of algorithms, including ID3 and C4.5.
- Because DT employ the "divide and conquer" strategy, they typically perform best when just a small number of highly significant qualities are present [6].
- Any modification to the training data leads to a modification in the attribute choices, which has an impact on the entire tree [11].
- The output properties also need to be singular and categorical [11].

1.7 Random Forest (RF)

1.7.1 Introduction to Random Forest

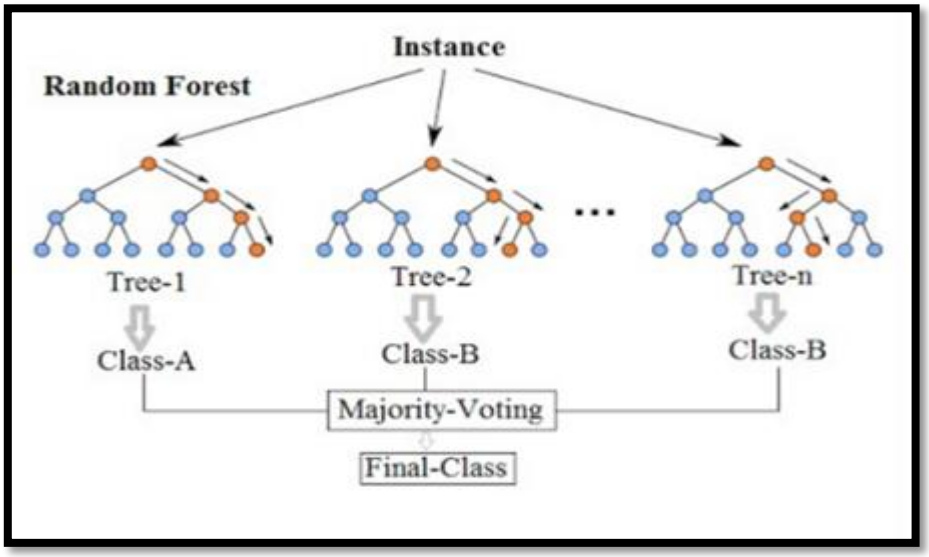
RF stands out as a potent ensemble ML algorithm. It operates by constructing numerous DT and amalgamating the results produced by each tree through a voting mechanism. The final output is determined by the majority of votes from the individual DT (Trevor Hastie, 2013) [41].

RF constructs multiple DT based on randomly selected subsets of attributes during training. The algorithm outputs the class that receives the most votes from the ensemble of trees. The relationship between accuracy and the number of trees exists, with a higher number of trees expected to yield a higher accuracy rate. Addressing a significant challenge in ML, overfitting occurs when a model excessively tailors itself to the training data, performing well only when tested on that specific dataset. RF mitigates this problem by aggregating votes from the ensemble of trees, preventing the model from overfitting and enhancing its generalization to new, unseen instances beyond the training data.

By considering the votes from each DT it creates, RF aims to attain significantly higher accuracy compared to a single DT [37].

Figure 1.5

Random Forest Simplified

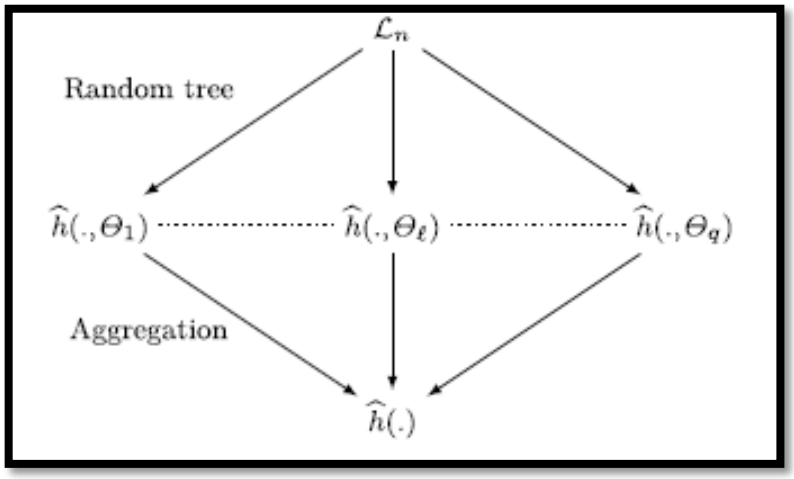


The general definition of RF, given by Breiman (2001), is as follows:

Let ($\hat{h}$ *(.,* Θ_1*)*,..., $\hat{h}$ *(.,*Θ_q*)* be a collection of tree predictors, withΘ_1,..., Θ_q q i.i.d. random variables independent of L_n. The RF predictor , $\hat{h}_{RF}$ is obtained by aggregating this collection of random trees. The aggregation is done as follows:

Figure 1.6

General scheme of Random Forest



- $\hat{h}_{RF}$ (x) = $\frac{1}{q}\sum_{l=1}^{q}\hat{h}$) (average of individual tree predictions) in regression.
- $\hat{h}_{RF}$ (x) = $\overset{argmax}{1\leq c\leq C}\sum_{l=1}^{q} 1_{\hat{h}(x,\Theta_l)=c}$ (majority vote among individual tree predictions) in classification.

The concept is depicted in the diagram shown in Figure 1.6. To achieve effective predictive performance, a RF method needs to assemble a collection of trees that possesses two key characteristics:

1. Diversity: The collection should be as diverse as possible. Aggregating predictors that are highly similar would result in little improvement, essentially creating a predictor akin to the individual ones.
2. Individual Predictive Capacity: The ensemble must consist of individual predictors with reasonable predictive capability. If all trees yield poor predictions for a new observation x, the aggregated predictions stand little chance of being accurate [56].

1.7.2 Random Forest Work

The RF method, also known as the Random Decision Forest, is a collaborative learning approach utilized for classification and diverse tasks. Its mechanism involves the construction of a collection of DTs during the training phase, with the output determined by the mean prediction of individual trees. Notably, in RF, there exists a direct proportionality between the number of trees in the forest and the resulting accuracy.

The RF algorithm unfolds in two main steps:

1. Creation of an RF Tree: This involves the following five stages:

- Selection of K random features from the total feature set m, where K is less than m.
- Determination of node d within the selected features using the best split point.
- Distribution of these nodes into daughter nodes through the best split.
- Repetition of the first three steps until l nodes are obtained.
- Iteration of the above steps n times to achieve p number of trees, where n is not equal top.

2. Data Classification based on an RF Tree: This involves the subsequent stages:

- Utilization of rules generated for each randomly formulated DT to classify data with test features.
- Calculation of votes for each target value.
- Considering the target with the highest votes as the final result of the RF algorithm [40].

1.7.3 Feature selection with Random Forests

Leveraging RF for feature selection is a valuable technique that enhances the performance of ML models. RF offer a feature importance metric, aiding in the identification of the most relevant features for a given task.

In addition to its well-established accuracy improvements, RF are widely adopted due to their ability to provide feature importance measures. These measures, including Overall, Permutation, and Tree Interpreter feature importance, contribute to the algorithm's popularity.

The calculation of overall feature importance involves assessing the decrease in node impurity, weighted by the probability of reaching that specific node. The rationale is that as we delve deeper into tree levels, the node impurity should decrease. This allows for an objective quantification of a node's impact through the drop in impurity. Gini impurity is computed for each node, where the node probability is determined by the number of samples reaching the node divided by the total number. Higher values indicate more crucial features. The process of determining Overall feature importance begins with:

1. Calculating nodes importance n_j of node j for every DT, using the following equation:

$$n_j = W_j\ C_j - W_{left(j)}\ C_{left(j)} - W_{right(j)}\ C_{right(j)} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (7)$$

Where:

- W_j is the probability of reaching node j,
- C_j is Gini impurity of the node j.
- $W_{left(j)}$and $W_{right(j)}$ are the probabilities of reaching the left and right child nodes of node j, respectively,
- $C_{left(j)}$ and $C_{right(j)}$ are the Gini impurities of the left and right child nodes of node j, respectively.

2. The importance of each feature (F) in the tree is calculated using Eq. 8 where m is total number of nodes:

$$F(j) = \frac{n_j}{\sum_{i=1}^{m} n_i} \text{ .....(8)}$$

3. The importance of each feature in RF (collection of k Trees) is calculated using the following equation [4]:

$$\text{Feature Importance } (i) = \frac{\sum_{j=1}^{m} F(j)}{k} \text{ ..... (9)}$$

1.7.4 Advantages and Disadvantages of Random Forest

Like any algorithm, the RF method comes with its own set of advantages and disadvantages. The pros and cons for RF we'll be explore in the following sections, related to using it in classification.

- **Advantages of Random Forest:**

1. Related to Accuracy: RF have high accuracy in both classification and regression tasks. The ensemble nature of the model helps in reducing overfitting and effectively capturing complicated patterns in the data.
2. To avoid Overfitting: The ensemble of DT, each trained on a distinct subset of the data, plays a pivotal role in mitigating overfitting. This will give stronger model, particularly in handling several datasets.
3. To avoid Outliers: RF exhibit robustness to outliers and noisy data, courtesy of their ensemble nature. the effect of Outlier will be less than other models, this is reflected on the overall model, contributing to increased model resilience [9].
4. To handling large number of input variables: RF appear proficiency in handling a large number of input variables without the need for variable omission.
5. In Variance Reduction: This model superior over other in reducing variance without a simultaneous increase in bias, contributing to more stable, and fixed predictions [49].

- **Disadvantages of Random Forest:**

1. Computational Complexity: One of the notable limitations of RF is their computational complexity. The model's prediction speed is often slower due to the involvement of multiple DT. Each tree within the forest independently processes the same input data to generate a prediction, followed by an aggregation mechanism, such as voting, to produce the final output. This sequential and parallel processing

increases the overall computational demand, making the prediction process more time-intensive.

2. Interpretability Challenge: In contrast to a single DT, the interpretability of RF is significantly more complex. While individual DT provide a transparent and intuitive structure that facilitates easy understanding and prediction, the ensemble nature of RF, which combines the outputs of numerous trees, introduces a layer of complexity. This aggregation obscures the direct interpretability of the model, making it harder to trace the decision-making process [51].

Recognizing these aspects, including both strengths and limitations, is essential for determining the suitability of the RF algorithm. This decision should be guided by the specific nature of the dataset and the goals of the ML application.

1.8 Literature Review and Related Work

This section provides a critical review of recent studies on DT, RF, jet classification, and the practical applications of DT algorithms, which are foundational to the current research.

Singh et al. (2019) conducted an extensive analysis of supervised classification mechanisms, which are widely regarded as essential tools for decision-making in ML. Classification refers to the systematic process of grouping data into distinct categories based on shared attributes. This approach has been applied across various fields, including bioinformatics, fraud detection, and banking. DT algorithms, often used in inductive learning, have been instrumental in developing classifiers that improve the reliability of classification processes. Naïve Bayes, a probabilistic induction method, is frequently employed due to its robustness against noise and superior performance compared to other inductive techniques [50]. While Singh et al. (2019) explore generalized supervised classification across multiple domains, the current research focuses on their application in HEP, addressing domain-specific challenges such as complex data structures and intricate jet interactions, thereby contributing uniquely to particle physics through specialized jet classification.

The versatility of DT is evident in their applications across biological sciences. Alqahtani et al. (2021) investigated the misclassification of beta coronavirus genomes, aiming to improve species identification accuracy. Their study demonstrated that amino

acid bias is a critical feature for classifying beta coronavirus species, achieving a remarkable accuracy rate of 99%. The findings highlight the utility of amino acid frequency as a reliable feature for categorization [56]. While Alqahtani et al. (2021) focus on biological classification, the present study extends these methodologies to HEP, tackling the unique challenges of jet classification in particle collisions. This contrast underscores the adaptability of DT algorithms across disparate scientific disciplines.

Moreno et al. (2019) employed physics-based sequential rearrangement algorithms to integrate data from multiple detector components, reconstructing particle jets based on the LHC. Their approach relies on jet identification algorithms analyze the collective properties of particles within a cascade to determine the characteristics of the initiating particle [39]. While Moreno et al. (2019) emphasize physics-based algorithms and detector configurations for jet identification, the current research leverages ML techniques to classify jets into distinct categories, offering a complementary approach to jet analysis in HEP. This distinction highlights complementary approaches to addressing challenges in jet identification, with our research providing a data-driven perspective leveraging advanced ML tools.

Duarte et al. (2018) drew on recent data from the Large Hadron Collider (LHC) to show that real-time event processing techniques boost physics capabilities. They developed a compiler package based on High-Level Synthesis (HLS), known as hls4ml, to deploy ML models on FPGAs. HLS allows for a significant decrease in firmware creation time and increases accessibility for a wide range of users. To see whether challenges in particle physics would benefit from employing FPGAs to conduct NN inference, they plot FPGA resource utilization and latency versus NN hyper parameters. They were able to fit our example jet substructure model into the available resources of modern FPGAs with a latency of 100 ns [14]. Duarte et al. (2018) focuses on hardware-accelerated deep learning for real-time processing in HEP, emphasizing FPGA integration. In contrast, our work applies interpretable ML models, specifically DT and RF, to classify jets based on their physical characteristics, with an emphasis on enhancing particle physics analyses through traditional ML metrics. This distinction highlights complementary yet distinct contributions to advancing ML applications in HEP.

Zenghui Wangb et al. (2019) confirms that ML is being used in a variety of industries, including retail, banking, education, and healthcare. Researchers are creating numerous algorithms employing knowledge of existing algorithms and skills from other domains to process the vast amounts of data coming from diverse sectors. Classification Trees, or CART Classification Trees, are among the ML DT methods that are highly potent. In this study, ID3 and C4.5—which are frequently utilized in classification issues—are the main subject. Ross Quinlan created an updated version of ID3 called C4.5. The accuracy of these algorithms' predictions is crucial. In this essay, an extensive evaluation of pertinent studies that sought to enhance the effectiveness of the algorithms and the various techniques employed was undertaken in this paper to study the prediction performance of DT algorithms. The various tree-based algorithms are also compared. Since there is no literature that has compiled pertinent advancements of DT - based algorithms, the main contribution of this review is to inform researchers of the advancements made to date and, in the end, to establish the groundwork for further study and advancements [38]. While Zenghui Wang et al. (2019) provides a general overview of DT advancements across multiple industries, our work is highly specialized, focusing on applying DT and RF models to classify particle jets in HEP. This specialization allows our research to address domain-specific challenges, such as capturing intricate particle interactions, which are not covered in the reviewed literature.

A study by Sohrab Zendehboudi et al. (2020) explored the classification of breast cancer using the Wisconsin Breast Cancer Database (WBCD), focusing on the application and effectiveness of ML and data mining techniques. The researchers introduced new methodologies for breast cancer classification by employing RF and Extremely Randomized Trees (Extra Trees, ET) algorithms. These approaches utilize DT as precise classifiers to achieve final classification outcomes. Both RF and ET methods involve four key stages: input identification, optimization of tree quantity, vote analysis, and final decision-making. Error analysis revealed that the RF and ET models produce clear results and outperform previously discussed tools and models in classifying the WBCD dataset. The relative importance of features was found to correlate with cell size uniformity and mitotic count. The study anticipates increased adoption of RF and ET algorithms in medical and healthcare systems, particularly for screening and diagnostic applications [20]. While Zendehboudi et al. (2020) explores DT-based ensemble learning for medical diagnostics, our work applies similar algorithms to a specialized

domain in physics, showcasing their utility in vastly different fields. The primary distinction lies in the dataset, features, and specific challenges of each domain medical diagnostics vs. HEP underscoring the versatility of DT-based models across disciplines.

The literature reviewed in this study showcases a wide array of methods used for particle classification in high-energy physics (HEP), spanning from traditional physics-informed algorithms to more recent deep learning models. Despite the impressive capabilities of these advanced approaches—especially in modeling complex jet structures—this research focuses on using Decision Trees (DT) and Random Forests (RF). These models were chosen for their interpretability, computational efficiency, and proven track record with structured datasets.

The choice to prioritize DT and RF over more sophisticated techniques such as Deep Neural Networks (DNNs), Support Vector Machines (SVMs), and Gradient Boosting Machines (GBMs) stems from a careful evaluation of trade-offs. For example, SVMs showed limitations in handling large-scale, multiclass problems, while GBMs demanded more computational resources and fine-tuning. In contrast, Random Forests struck a practical balance—offering strong performance with less complexity and minimal data preprocessing.

This methodological direction is supported by prior studies like those by Zendehboudi et al. (2020) and Singh et al. (2019), which highlight the effectiveness of tree-based models in classification tasks across different fields. While deep learning remains an exciting frontier—especially when enhanced by powerful computing and interpretability tools—the current study builds on a solid, transparent foundation using ensemble methods grounded in decision trees.

Chapter Two

Methodology

The success of any research endeavor relies heavily on the chosen methodology, the systematic approach used to answer the study questions and achieve the objectives. In this chapter, First, we will discuss how the data was collected and processed. Second, we provide a comprehensive overview of the methodology employed in our research, i.e., DT and RF, in terms of training model, optimization, evaluation model. and offering a detailed roadmap of how we approached, executed, and analyzed our study. Lastly, we will introduce the environment used, Sklearn for implementing our study.

2.1 Research Design

This research is of the quantitative experimental type, relying on the analysis of collected data to classify jets using 53 key features. The variables used include basic kinematic features such as particle tangential momentum and non-decomposable mass, in addition to N-subjettiness variables using different beta values (1 and 2) and tau3/tau2 ratios. Energy correlation functions (ECFs) with different beta values, modified Mass Drop Tagger (mMDT) variables, and mass variables. The objective is to build and evaluate machine learning models specifically, DT and RF for the classification of jets based on their physical characteristics.

2.2 The Dataset

2.2.1 Data Sources

The research experiment will be carried out in this study with the use of a data set provided by the Zenodo platform collisions pT= 1TeV simulated jets formed from light quarks q, gluons g, W and Z bosons, and top quarks created in √s= 13TeV proton-proton collisions.

The data set was prepared using an LHC detector's configuration and parametric description, each jet is represented by a feature vector comprising 53 high-level features (HLFs). These features encapsulate the jet’s kinematic (e.g., momentum, pseudorapidity), structural (e.g., N-subjettiness, energy correlation functions), and statistical properties. These variables are specifically designed to aid in distinguishing between different types of jets and serve as input features for the machine learning

models. Furthermore, the dataset includes classification labels that denote the true origin of each jet, enabling supervised learning and model evaluation [18].

Dataset of high-pT jets from simulations of LHC proton-proton collisions prepared for Fast ML Lab (FastML Lab) /HLS4ML studies. Fast ML Lab is a multidisciplinary research group of physicists, engineers, and computer scientists with the interest of exploring the use of ML algorithms for some exclusive and challenging scientific applications. Their focal projects consist of real-time, on-detector, and low-latency ML applications on one side and high-throughput heterogeneous computing big-data challenges on the other. They look forward to deploying complex ML algorithms for the advancement of fundamental physics exploration-from the world's largest colliders to the most intense particle beams to the cosmos.

2.2.2 Data loading and Pre-Processing

We started by downloading the data set from Zenodo platform then we imported a few libraries, the others will be imported as and when they are used in the program at different stages. For now, we imported the libraries which will help me in importing and preparing the dataset for training and testing the model.

We then import our dataset to Jupyter, which is stored in the.h5 file. This is done using the h5py library. In our dataset the data has been processed whenever less than 150 particles are reconstructed, the list is filled with zeros.

```
import h5py
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

df='C:/Users/hp/Desktop/marwathesis/train/jetImage_0_150p_20000_30000.h5'
with h5py.File(df, "r") as f:
# List all groups
print("Keys: %s" % f.keys())
a_group_key = list(f.keys())[5]

# Get the train data jets
y = list(f[a_group_key])
```

The data contains a set of lists describing the jets to be classified, which are:

['jetConstituentList', 'jetFeatureNames', 'jetImage', 'jetImageECAL', 'jetImageHCAL', 'jets', 'particleFeatureNames']

We used 'jets' lists to classify jets that contains HLFs used for jet identification in particle physics research. These features are extracted from jet events and are used in ML models to classify and analyze the jets.

- Advantages of Using HLFs
  Simplified Data: HLFs reduce the complexity of the data by summarizing important characteristics of the jets, making it easier to train and deploy ML models.

- Numerous research investigations have demonstrated that HLFs exhibit significant efficacy in classification tasks, frequently attaining superior performance metrics. These features are adept at encapsulating the essential discriminative information required to differentiate between various categories of jets.

- Computational Efficiency: Using HLFs can significantly reduce the computational resources required compared to processing large volumes of particle-level data.

The 'jets' is a list of two-dimensional lists (10000,59). In the context of jet classification, the last six digits in the 'jets' are reserved for the classification labels of the jet. The provided labels classify the jet according to its specific type or category, determined through the analysis of extracted features and characteristics derived from the dataset.

```
X= [] # Initialize an empty list to hold the jet information
# Iterate over each row in the data
for i in range (10000):
    # Append jet information (except for the last six digits) to X
    X.append(y[i][0:53])

Y = [] # Initialize an empty list to hold the classification labels


# Iterate over each row in the data
for i in range (10000):
    # Iterate over the last six digits in the 'jets' data to find the
classification labels
    for j in range (6):
        if y[i][53 + j] == 1:  # Check if the classification label is 1
           Y.append(j)  # Append the index of the classification label to Y
```

I Will now imports train_test_split to split our datasets into training and testing datasets. Then, I'll import decision tree classifier and random forest classifier that I will be using to train and test the data.

We divided the dataset into two separate groups (train data and test data).

```
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.20,
random_state=42)
```

To evaluate model performance reliably, we divided the dataset into training and testing sets, with 80% used for training and 20% for testing. This split ratio is commonly adopted as it provides a sufficient amount of data for model learning while reserving a reasonable portion for unbiased evaluation. Although this method is effective in our experiments, other strategies such as cross-validation can be applied, especially in problems where data is limited. Cross-validation mitigates the risk of overfitting to a particular train and test split by rotating the evaluation across multiple subsets.

2.3 Decision Tree and Random Forest

DT and RF are types of ML algorithms commonly used in artificial intelligence for their ability to handle complex classification and regression tasks with interpretability and ease. DT work by recursively splitting the dataset into subsets based on the most significant attribute at each node, thus forming a tree-like structure where each branch represents a decision rule. In ML, DT operate by autonomously identifying optimal splits through iterative processes, guided by criteria such as information gain or Gini impurity, eliminating the need for explicit rule definition by the programmer. RF improve upon DT by generating multiple trees trained on random subsets of the training data and aggregating their predictions, thereby reducing overfitting and enhancing generalization capabilities.

The choice of DT and RF in this study is grounded in several factors. First, both algorithms are utilized to analyze jet properties, capitalizing on their proficiency in managing high-dimensional data and capturing intricate feature interactions. Second, they are computationally efficient and capable of handling a large number of input features—such as the 53 high-level features (HLFs) used to describe jets in the dataset. By incorporating attributes such as total energy and particle count, derived from Zenodo datasets, these models effectively classify jets into categories including light quarks (q), gluons (g), W and Z bosons, and top quarks.

The adaptability and robustness of DT and RF render them essential tools for deciphering complex relationships within HEP data and delivering reliable predictions

based on observed characteristics. Although DT were introduced in the 1960s and are relatively simple in structure, their integration into RF in the 2000s has significantly expanded their applicability, achieving notable accuracy and efficiency across diverse domains such as finance, healthcare, marketing, and scientific research [24].

Alternative model such as NNs was considered but ultimately not selected. NNs, while powerful in capturing complex nonlinear relationships, require significantly more tuning, longer training times, and are often considered "black-box" models with limited interpretability. On the other hand, DT and RF also help discover meaningful relationships between features. Understanding these relationships is important, as some features may be ineffective when used alone, but become powerful when combined with other features. Therefore, the DT and RF provide a balanced trade-off between accuracy, interpretability, and ease of implementation, making them particularly suitable for the classification of jets based on physical properties derived from collider simulations.

2.3.1 Model Training

To support the advancement of experimental methodologies in HEP, this section outlines the methodology for training DT and RF. The construction, optimization, and evaluation of the model involve multiple interconnected phases, which collectively contribute to the development of a predictive training model for identifying particle types in HEP experiments.

Unlike linear models, which presume that the data is linear, DT make very few assumptions about the training set. If the tree structure is allowed to grow unrestrained, it will fit the training data very closely possibly overfitting it. A model like this is commonly referred to as nonparametric; this is not because it lacks parameters it frequently has a large number but rather because the number of parameters is unknown before training, allowing the model structure to closely resemble the data. A parametric model, on the other hand, like a linear model, has a fixed number of parameters, which limits its degree of freedom and lowers the chance of overfitting (but raises the danger of underfitting).

You must limit the DT freedom during training in order to prevent overfitting the training set. Using DT requires adjusting several parameters, or "hyperparameters," to

maximize the model's performance. Thorough adjustment of these factors is crucial when predicting particle kinds in HEP [19]. Finding the parameters and parametrizing a structure resembling a tree is the current challenge. Recursively splitting nodes and creating leaves is the process of creating a structure resembling a tree. In scikit-learn, DT have several parameters that can be tuned to optimize their performance. Here's an explanation of some of the key parameters:

- Criterion: use a parameter (such as split position within the feature set) to divide the criterion. Therefore, one of the DT parameters can be a threshold for the feature values. There are two choices available to us: Entropy and Gini impurity are typical criteria. The option affects the DT ability to forecast particle kinds accurately by influencing how it assesses the goodness of splits. The Entropy was chosen as the splitting criterion due to its efficiency in evaluating the quality of a split.
- Splitter Function: This parameter determines the strategy employed for selecting splits at each node. The "best" option identifies the optimal split, while "random" selects the best split from a random subset. The "best" option was used to allow the model to select the most optimal split at each node.
- Tree Depth (Max Depth): This parameter controls the depth of the DT by limiting the number of levels. We tested depths in the range of 2 to 20 to prevent overfitting, particularly in HEP applications where datasets often contain significant complexity and noise. Selecting an appropriate depth ensures the model generalizes effectively to unseen data.
- Minimum Samples Split: This defines the smallest number of samples necessary to split an internal node. A range from 50 to 150 was tested. Increasing this value can help mitigate overfitting by restricting the tree's growth.
- Minimum Samples Leaf: This specifies the minimum number of samples required to form a leaf node. A range from 10 to 100 was tested to ensure leaf nodes had a sufficient number of samples, promoting model stability and robustness.
- Maximum Features: This parameter dictates the number of features evaluated when identifying the best split. Options include "auto" (where max features = √n_features), "sqrt" (square root of the total features), and "log2" (logarithm base 2 of the total features). It can also be defined as an integer or float value.

In conclusion, the effective parametrization of DT for particle type prediction in HEP necessitates meticulous tuning of hyperparameters. This process ensures a balance between model complexity and generalization capability. The optimal hyperparameters for a RF model in this domain vary depending on the dataset's characteristics and the specific objectives of the analysis. The process of hyperparameter tuning frequently involves the use of methods like randomized or grid search.

A potent ensemble learning technique called RF creates several DT during training and outputs the mode of each tree's predictions (for classification issues). For the implementation of the RF algorithm, there are several hyper-parameters in scikit-learn that you might consider tuning [55]:

- Number of Trees (n_estimators): The number of DT in the ensemble is set by this hyperparameter. Performance is usually enhanced by adding more trees, but doing so has a computational cost. A range of 10 to 400 trees was tested to ensure stability and robustness of predictions.
- Tree Depth (max_depth): Similar to DT, used to control the depth of each tree and avoid overfitting. Depth values between 5 and 40 were explored to control model complexity and improve generalization.
- Minimum Samples Split (min_samples_split): This parameter specifies the minimum number of samples required to split an internal node. By preventing splits that result in excessively small subsets, it helps regulate tree growth and reduces the risk of overfitting. Values from 5 to 20 were tested to regulate the growth of individual trees and prevent overly specific splits.
- Minimum Samples Leaf (min_samples_leaf): This parameter determines the minimum number of samples required to form a leaf node. A range of 2 to 50 was tested to ensure each leaf node contained enough samples, helping the model avoid capturing noise in the data.

A Systematic Approach to Building and Training DT and RF Models

1. Downloading the Data from Zenodo Platform:

Obtain the dataset from the Zenodo platform, ensuring it is suitable for the intended classification task.

2. **Loading and Preparing the Data:**

Load the dataset and perform necessary preprocessing steps, such as handling missing values, encoding categorical variables, and scaling features, to prepare the data for training and testing.

3. **Building the Model:**

Utilize the DecisionTreeClassifier and RandomForestClassifier classes from the scikit-learn library to construct models capable of classifying data into five distinct categories.

4. **Training the Model and Evaluating it:**

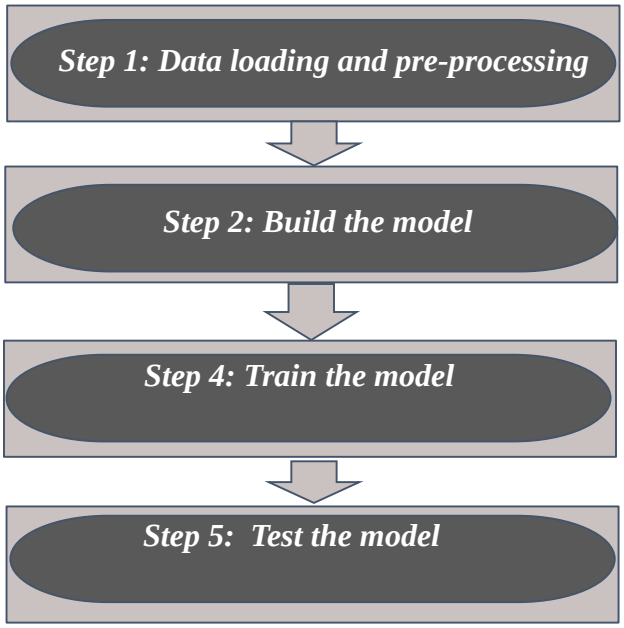
- Train the DT and RF models using the preprocessed data.
- The splitting criterion (e.g., Gini impurity or entropy for DT), maximum tree depth (max_depth), minimum samples needed for splitting (min_samples_split), number of trees in the forest (n_estimators for RF), and maximum number of features taken into consideration for splitting (max_features) are among the important hyperparameters that should be optimized.
- Assess model performance on a held-out test set to evaluate classification accuracy and identify potential areas for improvement.

5. **Improving Performance:**

- Analyze evaluation metrics to refine the models by adjusting hyperparameters or incorporating feature selection techniques.
- Iterate the training and evaluation process, incorporating modifications to achieve optimal classification accuracy.

Figure 2.1

Flowchart of a machine learning model



2.3.2 Development and Training of DT and RF Models for Jet Classification Utilizing the Scikit-learn (Sklearn) Framework

This section outlines the implementation of Python-based code for constructing and training DT and RF models aimed at jet classification. The primary goal is to illustrate the application of ML techniques in categorizing jets according to their distinct features. The scikit-learn library is employed to create and train these models using the provided dataset.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
classification report
import numpy as np
# Load and preprocess your data (Y, X)
Y = np.array(Y)
X = np.array(X)
# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)
# Build the model
clf_model =
DecisionTreeClassifier(criterion="entropy",splitter="best",max_depth=8,min_samples_s
plit=50,min_samples_leaf=20,max_features=40)
clf_model=
RandomForestClassifier(n_estimators=100,criterion="entropy",max_depth=10,min_samples
 split=5,min samples_leaf=4)
# Train the model
clf_model.fit(x_train,y_train)
# Evaluate the model
y_pred = clf_model.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score (y_test, y_pred, average='weighted')
results= {}
# Store results
results= {'Accuracy': accuracy, 'Precision': precision, 'Recall': recall, 'F1-
score': f1}

# Print or log classification report for detailed evaluation
print("Classification Report:")
print(classification_report(y_test, y_pred))
print(results)
```

1. Importing Libraries: The code imports `DecisionTreeClassifier`, `Random Foret Classifier`from Sklearn, along with other necessary libraries such as NumPy and `sklearn.metrics` for data manipulation and model evaluation.
2. Data Preparation: The code snippet assumes that you have preprocessed your data and split it into training and testing sets using `train_test_split` from `sklearn.model_selection`. It also converts the data into NumPy arrays.
3. Model Definition: The DT model is defined with: `criterion="entropy"`: uses entropy as the splitting criterion, `splitter="best"`: chooses the best split at each node, `max_depth=8`: limits the tree depth to prevent overfitting, `min_samples_split=50`: minimum samples required to split a node, `min_samples_leaf=20`: minimum samples required at leaf nodes, `max_features=40`: uses only 40 features for splitting. And The Random Foret model is defined with: `criterion="entropy"`, number of trees = 100, the maximum depth = 10, minimum samples split = 5 and minimum samples leaf = 4.
4. Model Training: The `model.fit` function is called to train the model on the training data (`x_train `and `y_train`). where the model learns patterns from training data.

5. Model Evaluation: Finally, `model. predict` is called to evaluate the trained model's performance on the testing data (`x_test` and `y_test`) and performance metrics are calculated.
6. Displaying Results: A classification report is printed using `classification_report`.

2.3.3 Optimization

The process of identifying the optimal model configuration or parameter set to achieve desired performance metrics is known as optimization. The primary objective is to enhance the model's predictive accuracy while mitigating risks of overfitting or underfitting. Among the various optimization techniques, feature importance stands out as a widely utilized method in ML for both feature selection and model interpretability. Feature importance involves assessing and ranking the contribution of each feature to the model's predictive outcomes. Commonly, algorithms such as DT or RF are employed to quantify feature importance, with metrics derived from each feature's role in reducing specific criteria (e.g., Gini impurity or entropy) during split decisions. While feature importance is computationally efficient and aids in data understanding, it may yield misleading results in cases of feature correlation or excessive model complexity. In this study, feature importance will be utilized to identify and select the most impactful features, thereby reducing dataset dimensionality and enhancing model performance for particle type prediction. This strategy focuses on prioritizing the most significant attributes, improving interpretability, and potentially minimizing overfitting risks.

```
# Get feature importances
importances = clf_model.feature_importances_
# Sort features by importance
indices = np.argsort(importances)[::-1]
# Display sorted feature importance
print ("Feature ranking:")
for f in range(x_train.shape[1]):
    print (f"{f + 1}. Feature {indices[f]} ({importances[indices[f]]})")
```

2.3.4 Model Evaluation

Assessing the performance and reliability of a model for predicting particle types in HEP requires a systematic evaluation process as a critical initial step. Various metrics and methodologies are employed to ensure the model's effectiveness when generalizing to new, unseen data. The model is developed through a classification approach, utilizing

a labeled dataset where each entry in the training set is associated with a specific class label. This process ensures the model's ability to accurately categorize data based on learned patterns. Afterwards, the model is applied to forecast the class labels of fresh or unseen data. The capacity of a classification model to accurately anticipate the class label of previously unknown data is known as predictive accuracy. Confusion matrix, total, per-class, recall, and precision are popular measures used to assess the accuracy of classification models. To facilitate the calculation of all other measures, a confusion matrix is first generated.

- **Confusion matrix**

The confusion matrix provides a thorough analysis of the performance for each class, breaking down the right and wrong guesses. By contrasting the expected and actual values of the outcomes, the performance is evaluated. As seen in Table 2.1, the data is tabulated as a confusion matrix [54].

Table 2.1

Confusion Matrix

		Actual	
		Positive	Negative
Predicted class	Positive	True Positive count (TP)	False Positive count (FP)
	Negative	False Negative count (FN)	True Negative count (TN)

Where:

1. True Positives (TP): Instances where the model correctly predicts the positive class (correctly identifies a particle type).
2. True Negatives (TN): Instances where the model correctly predicts the negative class (correctly identifies a non-particle type).
3. False Positives (FP): Instances where the model incorrectly predicts the positive class (predicts a particle type when it is not).
4. False Negatives (FN): Instances where the model incorrectly predicts the negative class (fails to identify a particle type when it is present).

- **Overall Accuracy**

The rate of accurate predictions made by the model is known as overall classifier accuracy. It calculates the percentage of all events in the dataset that were accurately predicted, including true positives and true negatives. The accuracy is calculated using the formula:

$$\text{Accuracy} = \frac{Numberofcorrectprediction}{Totalnumberofprediction}$$

$$= \frac{TP+TN}{TP+TN+FP+FN} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (10)$$

- **Precision**

As the sum of correctly categorized majority class values (True positives) and erroneously classified majority class values (False positives) divided by the ratio of correctly classified majority class values (True positives) is known as precision. It ought to be elevated.

$$\text{Precision} = \frac{TP}{TP+FP} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (11)$$

- **Recall**

As the total of correctly classified majority class values (True positives) and erroneously classified minority class values (False Negatives) divided by the ratio of correctly classified majority class values (True positives) is known as recall. Recall calculates how accurate the classifier is at predicting the majority class. It ought to be elevated [54]:

$$\text{Recall} = \frac{TP}{TP+FN} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (12)$$

Interpretation:

- High Accuracy: A high accuracy score means that, for the most part, the model is correctly predicting the data. It is an easy-to-understand statistic that is straightforward and intuitive.
- Limitations: A whole picture might not always be obtained from accuracy alone, particularly when there are unequal classes present. A model that predicts the

majority class for every case in the dataset can still achieve high accuracy, but it might not be useful if one class predominates in it.

- Consideration: For a more thorough assessment, depending on the sort of particle type prediction task, it's critical to take into account additional metrics including precision, recall, and the confusion matrix, particularly when working with unbalanced datasets.

- **F1-score**

It is the harmonic mean of precision and recall, defined as:

This metric is particularly important in imbalanced datasets, where accuracy alone can be misleading. The F1-score balances the trade-off between false positives and false negatives, making it a robust indicator of performance in detecting rare or less represented particle types.

The use of F1-score complements the confusion matrix analysis by summarizing the model's effectiveness in a single metric that penalizes extreme values in either precision or recall. This is especially relevant in high-energy physics, where both false discoveries and missed detections carry significant scientific implications.

These metrics were selected because they offer a balanced view of classifier performance, especially in multi-class and imbalanced settings typical in HEP research. They collectively allow the examination of both correctness and completeness in model predictions.

AUC-ROC (Area Under the Receiver Operating Characteristic Curve): This metric evaluates the model's ability to discriminate between classes at various threshold settings. Although not used in this study, AUC-ROC is particularly useful in binary classification tasks and could be extended to multi-class problems. Future studies may incorporate AUC-ROC to further enrich the evaluation framework, particularly in applications where class distributions are highly skewed.

2.4 Sklearn

A well-known open-source ML library for conventional ML is called Scikit-learn (Sklearn). It has techniques for clustering, regression, and classification. It enables us to deal with support vector machines in addition to other things. Because Scikit-learn is made to operate with NumPy and Matplotlib, our lives will be a lot easier [13].

Scikit-learn is a robust Python library that brings together an impressive collection of cutting-edge machine learning algorithms suitable for various medium-scale tasks, whether supervised or unsupervised. It was designed to make machine learning accessible to those who may not specialize in the field, using a user-friendly and versatile programming language. The library emphasizes clear documentation, a consistent API approach, strong performance, and overall ease of use. With its distribution under a simple BSD license and minimal dependency requirements, Scikit-learn is an excellent choice for both commercial and academic applications.

Scikit-learn satisfies the increasing demand for statistical data analysis by non-specialists in subjects other than computer science, such biology and physics, as well as in the software and online sectors.

There are several ways in which Scikit-learn is different from other Python ML toolboxes:

1. It is licensed under the BSD license. In contrast to MDP (Zito et al., 2008) and pybrain (Schaul et al., 2010)
2. It integrates compiled code for efficiency; in contrast to pymvpa (Hanke et al., 2009), which has optional dependencies like R and shogun
3. In addition, it focuses on imperative programming, as opposed to pybrain, which uses a data-flow framework.

The software package is predominantly coded in Python, yet it incorporates reference implementations of Support Vector Machines and generalized linear models, which are derived from the C++ libraries LibSVM (Chang and Lin, 2001) and LibLinear (Fan et al., 2008), under compatible licensing terms. Binary packages for this software are available across a variety of platforms, encompassing Windows and all POSIX-compliant systems [42].

2.5 Methodological Limitations

This study faced several methodological limitations that may have impacted the final results. The most notable of these limitations is the limited computing resources available. The experiments were run on a device with modest capabilities (an Intel i5-5200U processor and 8GB of RAM), which imposed constraints on the amount of data that could be processed at a time, impacting the speed of training and the ability to implement larger experiments.

The study was also run in a Python-based environment using Jupyter Notebook. This meant that the execution was carried out locally, without the use of cloud computing technologies or dedicated big data processing environments. This may limit scalability and the ability to replicate experiments more effectively.

The results achieved reflect the performance of the models in this environment and settings, and higher performance may be possible with a more advanced execution environment and greater computing resources.

Chapter Three

Results and Discussion

This chapter outlines the development, training, and evaluation of the DT model, along with the implementation of the RF model. It explores a methodology for comparing these algorithms in the context of classifying particle jets in HEP. Both models were trained and tested on a dataset containing particle attributes and their respective classifications, with the objective of assessing their effectiveness in accurately identifying these jets.

The chapter presents the results obtained for each model and includes a comparative analysis of their predictive performance. The DT model offers a simple and interpretable classification approach, whereas the RF model employs ensemble learning to achieve greater accuracy and robustness.

The subsequent chapter provides an in-depth examination of the methodology used for implementing, evaluating, and comparing the DT and RF models for particle jet classification. By detailing the steps involved in model training, parameter optimization, and performance metric evaluation, this chapter seeks to establish a comprehensive framework for researchers and practitioners in HEP who aim to utilize ML techniques for particle classification.

Through this investigation, the study aims to enhance the understanding of how ML can advance HEP, particularly in the classification of particle jets, while offering insights that may guide future research and practical applications in this field.

3.1 The Environment of Experiment

This section details the technical specifications of the equipment utilized for the development and testing of the model, along with the computational setup employed during the process.

1. Device Details

- Processor: Intel(R) Core (TM) i5-5200U CPU @ 2.20GHz 2.20 GHz
- Installed RAM: 8.00 GB (7.89 GB usable)
- System Type: 64-bit operating system, x64-based processor

2. Environment

The development and experimentation of the model are conducted using Jupyter Notebook, which operates on Python 3 as the primary interface.

3.2 Decision Tree (DT)

For DT, we started with the default set of parameters that we used in our research, as given in Table 3.1, and then adjusted only the first one in order to empirically determine the optimal values for the hyperparameters. We changed the second parameter after finding the optimal value for the first one, leaving the first one at the value we discovered in the prior iteration. Although it might have excluded certain potential combinations that would have worked better, iterating this procedure gave us a nice mix of hyperparameter values.

Table 3.1

A Summary of the Hyperparameters Used for DT

Hyperparameter	Description
Criterion Function	The function to measure the quality of a split (default: Gini).
Splitter Function	The strategy used to split at each node (default: Best).
Maximum Depth	Controls the maximum depth of the tree. The depth is the longest path from the root node to a leaf node (default: None).
Minimum Samples Split	The minimum number of samples required to split an internal node (default: 2).
Minimum Samples Leaf	The minimum number of samples required to be at a leaf node (default: 1).
Maximum Features	The number of features to consider when looking for the best split (default: None {uses all features}).

We have set `random_state=42` in `train_test_split` to ensure that the same 80% of the data is always used for training and the same 20% for testing, and we set `random_state=42` in `DecisionTreeClassifier` to ensure the same tree structure is built each time, this providing repeatable and clear results as shown in Figure 3.1.

Figure 3.1

Decision Tree with default parameters Code

```
from sklearn.tree import DecisionTreeClassifier
x_train,x_test,y_train,y_test= train_test_split(X,Y,test_size=0.20,random_state=42)
clf_model = DecisionTreeClassifier(random_state=42)
clf_model.fit(x_train,y_train)
y_predict = clf_model.predict(x_test)
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
accuracy_score(y_test,y_predict)
```

The DT with default parameters achieved an accuracy of 0.723. To enhance its performance, we will optimize it by tuning the hyperparameters.

Here are some hyperparameters specific to DT that are commonly tuned for this task:

- **Criterion Function**

In the previous chapter, we discussed the concept of the criterion function within DT, exploring its significance in determining optimal splits at each node during the tree construction process. Furthermore, we implemented a DT classifier using scikit-learn, experimenting with different criterion functions to observe their impact on model performance. By recording the accuracy scores attained with each criterion function in the classification task, we aimed to discern the effectiveness of various splitting criteria, as depicted in Table 3.2. Additionally, we e visualized the performance discrepancies among different criterion functions through graphical representations, facilitating a comprehensive understanding of their influence on DT classification outcomes, as illustrated in Figure 3.2.

Table 3.2

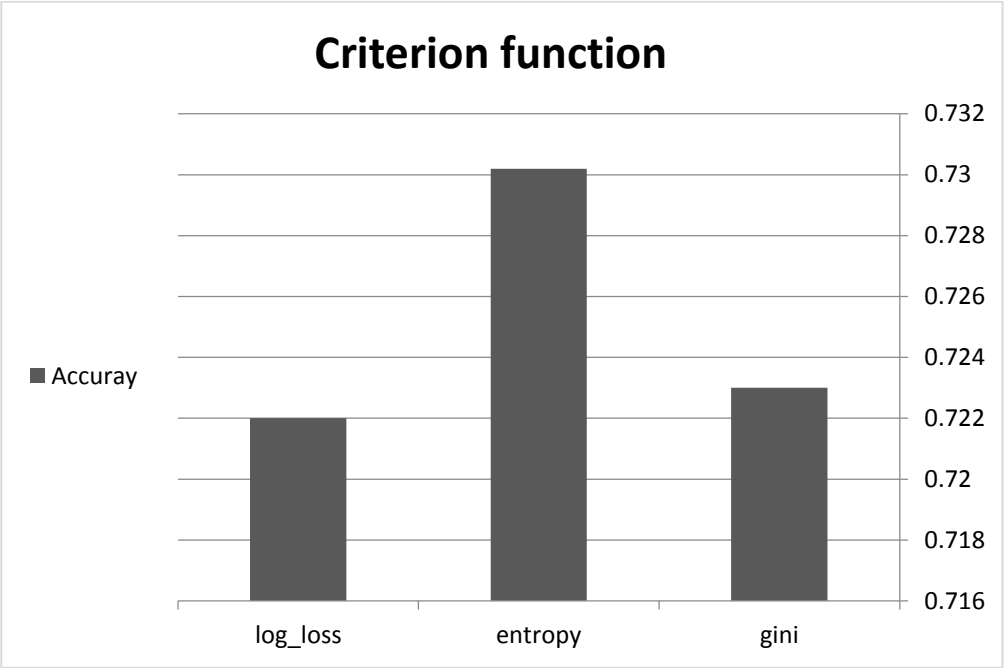
Accuracy of Different Criterion Functions of DT

Criterion Function	Accuracy
Gini	0.7230
Entropy	0.7302
log_loss	0.7220

As shown in the previous table, the best result was obtained when entropy function was used (0.7302).

Figure 3.2

Chart Accuracy of Different Criterion Functions of DT



To determine whether Entropy or Gini is better for building our model and achieving the highest possible accuracy in jet classification, we executed the code 20 times using different datasets then we calculated the average and variance of the results as shown in Table 3.3.

Table 3.3

Summary for Two Groups (Entropy vs. Gini) DT

SUMMARY				
Groups	*Count*	*Sum*	*Average*	*Variance*
Entropy	20	14.6045	0.730225	0.000073
Gini	20	14.4608	0.723040	0.000085

Here, the average accuracy for Entropy (0.730225) is higher than for Gini (0.723040). The Entropy group also has a slightly lower variance (0.000073) than the Gini group (0.000085), suggesting that the Entropy group exhibits less variability in its values compared to Gini.

In our research, we applied Anova: Single Factor as a statistical analysis method as shown in Table 3.4. This allowed us to investigate and analyze the variation among two

specific groups, namely Entropy and Gini. Through this analysis, we aimed to determine whether there were any statistically significant differences between these groups with respect to our chosen variables.

Table 3.4

Anova: Single Factor for Two Groups (Entropy vs. Gini) DT

ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	0.000516	1	0.000516	6.542502	0.014644	4.098172
Within Groups	0.002998	38	0.000079			
Total	0.003515	39				

The p-value and critical F-value help us determine whether the differences in accuracy between Entropy and Gini are significant or if they could have happened by chance. The ANOVA test further confirms that this difference is statistically significant, with an F-statistic of 6.54 and a P-value of 0.014, which is well below the 0.05 significance threshold. This indicates that the observed improvement with Entropy is unlikely to be due to random chance. Therefore, Entropy is the superior criterion for achieving higher classification accuracy in this multi-class jet classification problem.

- **Splitter Function**

The choice of splitter function in DT is problem-specific, and there isn't a one-size-fits-all solution. The most suitable splitter function can vary depending on the features of the dataset, the complexity of the problem, and other factors. It's crucial to experiment with different splitter functions and evaluate their performance empirically to determine the most effective one for your particular task. With this in mind, we conducted experiments using two different splitter functions, running experiment 20 times. We recorded the accuracy for each run and calculated the mean accuracy, as presented in Table 3.5. Based on the previous results, we used entropy criterion function, which gave me the best accuracy.

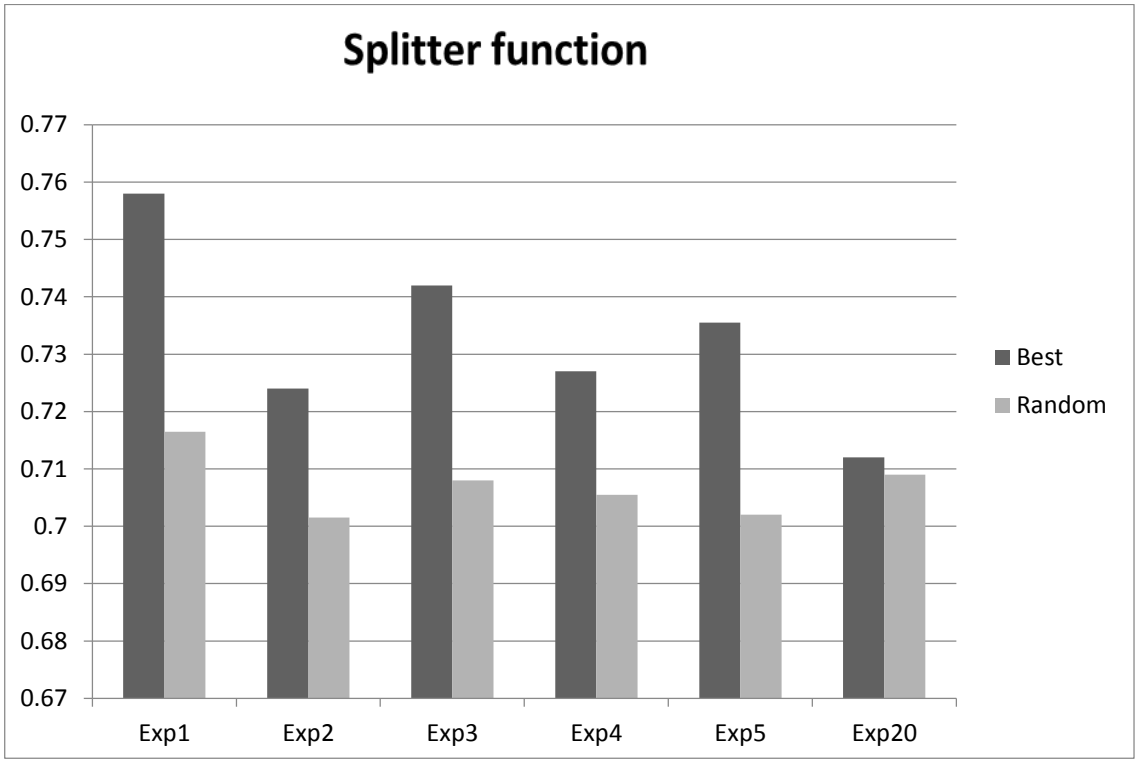
Table 3.5

Accuracy of Different Splitter Functions of DT

Splitter function	Exp1	Exp2	Exp3	Exp4	Exp5	…. Exp20	Mean Accuracy
Best	0.7580	0.7240	0.7420	0.7270	0.7355	0.712	0.7232
Random	0.7165	0.7015	0.7080	0.7055	0.7020	0.709	0.7035

Figure 3.3

Chart Accuracy of Different Splitter Functions of DT



The code has been executed 20 times, and we have consistently evaluated the accuracy of the highest accuracy splitter function to determine which ones repeatedly yielded the best results. Then we calculated the mean accuracy of each best and random function to improving the model architecture and selecting the appropriate splitter function to achieve the highest possible accuracy in jet classification. As shown in the table above, the splitter function which gave the best mean accuracy was best function (0.7232).

In order to analyze the result well, we must know the behavior of the DT using both the "best" and "random" splitter:

- In the best splitter, the algorithm evaluates all features and thresholds to determine the split that provides the best separation between classes. This method is computationally expensive but delivers superior accuracy.
- In the random splitter, randomness in feature and threshold selection reduces computation time. However, it compromises accuracy because the splits may not always optimally separate the jet categories.

The choice of the "best" splitter function is theoretically suitable for jet classification because it evaluates all potential splits and selects the one that maximizes information gain, ensuring that the most meaningful distinctions in the HLFs are captured. The characteristics of invariant mass and transverse momentum encapsulate intricate yet crucial patterns essential for distinguishing between quark and gluon jets. In a study titled "Quark jet versus gluon jet: fully-connected NNs with high-level features," the application of HLFs for classifying quark and gluon jets is investigated. The study utilized fully-connected NNs trained on physically meaningful variables, such as invariant mass and transverse momentum, to differentiate jets initiated by quarks from those initiated by gluons. The results indicate that these high-level features play a pivotal role in capturing the nuanced differences required for precise classification [31].

This implies that selecting an optimal splitting strategy in our research is more advantageous than relying on random splitting. An optimal splitter prioritizes precise divisions that improve generalization and accuracy, rendering it more suitable than a random splitter, which may fail to account for critical data relationships.

- **Maximum Depth**

Determining the maximum depth of a DT is a critical aspect that can significantly impact the model's performance and ability to generalize. The term "maximum depth" refers to the maximum number of levels or layers that the tree can have from the root node to the deepest leaf node. We used the same values of variables: criterion function: entropy and splitter: best. So, as shown in the table below, we have experimented with different values of maximum depth and recorded the accuracy of each in the Table 3.6.

Table 3.6

Accuracy of Varying Maximum Depth of DT

Maximum depth	Mean Test Accuracy
2	0.7339
4	0.7594
5	0.7702
6	0.7709
7	0.7751
8	0.7763
9	0.7632
10	0.7545
15	0.7230
20	0.7220

The results in Table 3.6, the mean accuracy over 20 experiments, show that the best accuracy obtained when the maximum depth = 8. Beyond this point, the accuracy begins to decline, with depths 9and 10 showing reduced performance (0.7632 and 0.7545, respectively). This decline suggests that the model begins to overfit the training data, losing its generalization capability on the test set.

The depth of a tree significantly impacts its complexity. When the depth increases, the model can become overly complex, leading to overfitting. Conversely, reducing the depth can simplify the model too much, potentially resulting in underfitting. Striking the right balance and determining the appropriate complexity level is crucial for optimal model performance. For this particular dataset, a maximum depth of 8 seems to hit that sweet spot between accuracy and generalization.

Underfitting (Shallow Trees): When a DT lacks sufficient depth, it may fail to capture the underlying patterns within the data effectively. This can result in high bias and low variance, leading to suboptimal accuracy on both training and test datasets. To address this, increasing the depth of the tree can enhance accuracy by enabling the model to identify more intricate patterns.

Overfitting (Deep Trees): Conversely, an excessively deep DT may overfit the training data by memorizing it rather than learning generalizable patterns. This scenario often leads to high variance and low bias, where the model performs well on the training data but poorly on unseen test data. Reducing the tree depth in such cases can improve generalization, thereby enhancing performance on new data.

These findings suggest that selecting an optimal splitting strategy in our study is more effective than relying on random splits. An optimal splitter prioritizes precise divisions that improve both generalization and accuracy, making it a more suitable choice compared to a random splitter, which may fail to capture critical relationships within the data Figure 3.4. –See Appendix A–

- **Minimum Samples Split**

This parameter determines the minimum number of samples necessary to divide an internal node within a DT. By establishing a threshold for the minimum number of samples required to split a node, the model can avoid overfitting, which often occurs when branches are formed with an insufficient number of samples. For instance, if the min_samples_split hyperparameter is configured to 20, the tree will only create a split if a node contains at least 20 samples. This means that an internal node must have at least 20 samples to be split. If certain internal node has only 15 samples, the node will not be allowed to split further, meaning the decision for classifying a new sample will be based on the majority of the class in that leaf node. It indirectly affects the tree's depth by preventing splits when the number of samples at a node is below the specified threshold. A small min_samples_split (e.g., 2) allows aggressive splitting, increasing the likelihood of overfitting as the tree grows very deep. In contrast, a larger min_samples_split constrains splitting, resulting in a shallower and less complex tree, which helps reduce overfitting but may cause underfitting if set too high. Proper tuning of both parameters is essential to strike a balance, ensuring the tree captures meaningful patterns without becoming overly complex or too simplistic.

To adjust both maximum depth and minimum samples split in a DT for our research, we used the best approach is to tune these parameters systematically using GridSearchCV method from sklearn. This allows us to test all combinations of minimum samples split values between (50-150) and maximum depth values between (4-8). We test all

combinations of the parameter grid (5 values of minimum samples split × 5 values of maximum depth = 25 combinations), and wrote the mean_test_score, it is the average accuracy across all folds for the combination.

Based on the previous results, we used the same values of variables: criterion function: entropy, splitter: best. We executed the code 20 times using different datasets as shown in Table 3.7.

Table 3.7

Accuracy of Varying Minimum Samples Split and Maximum Depth of DT

Dataset number	Minimum samples split	Maximum depth	Mean test score
1	50	8	0.7776
2	100	8	0.7758
3	50	8	0.7798
4	100	7	0.7717
5	50	6	0.7696
6	50	6	0.7811
7	50	7	0.7750
8	50	6	0.7761
9	75	8	0.7678
10	50	7	0.7748
11	50	8	0.7761
12	50	7	0.7761
13	75	7	0.7753
14	50	7	0.7606
15	50	7	0.7750
16	50	6	0.7761
17	75	8	0.7671
18	50	6	0.7706
19	125	8	0.7762
20	50	7	0.7681

The results from the 20 datasets can be interpreted to identify the best values for min_samples_split and max_depth that optimize accuracy. Among all combinations, the highest mean test score of 0.7811 occurs when min_samples_split = 50 and max_depth = 6 (Dataset 6). Additionally, other combinations with min_samples_split = 50 and varying max_depth values (such as 7 and 8) also yield competitive results close to the highest accuracy, such as 0.7798 (Dataset 3) and 0.7776 (Dataset 1).

To choose the best values, the focus should be on combinations that consistently produce high accuracy across datasets. So, we can study the case by using Heatmap graph to visualize the relationship between the mean test score, min_samples_split, and max_depth as shown in Figure 3.5 –See Appendix A–.

The heatmap above shows mean test scores for different combinations of max_depth and min_samples_split. The analysis of the results reveals that the optimal model performance is achieved with a max_depth of 8 and a min_samples_split of 50, yielding the highest mean test score of 0.7778. Across all configurations, max_depth = 8 consistently outperforms other depths (6 and 7), demonstrating its ability to provide a better fit to the data. On the other hand, lower values of min_samples_split (e.g., 50) generally produce higher scores, while increasing it to 75 or 125 leads to a slight drop in performance, likely due to underfitting. For instance, the lowest score of 0.7674 occurs at max_depth = 8 and min_samples_split = 75, highlighting that certain combinations can degrade model performance.

In summary, the results suggest that a max_depth of 8 and a min_samples_split of 50 provide the best balance for model performance.

- **Minimum Samples Leaf**

Similar to min samples split, but focuses on the leaf nodes. Setting this parameter appropriately ensures that each leaf represents a sufficient number of samples, promoting generalization and preventing the creation of overly specific rules. We used the same values of variables: criterion function: entropy, splitter: best, maximum depth: 8 and minimum samples split: 50.

We have experimented different values of minimum samples leaf and wrote the accuracy of each in the Table 3.8.

Table 3.8

Accuracy of Varying Minimum Samples Leaf of DT

Minimum samples leaf	Mean Accuracy
10	0.7781
20	0.7820
30	0.7769
40	0.7750
50	0.7744
60	0.7736
80	0.7701
100	0.7659

The results in Table 3.8, the mean accuracy over 20 experiments, show that the best accuracy obtained when the minimum samples leaf = 20.

Lower values, such as 10, resulted in an average accuracy of 0.7781; however, this may induce overfitting, as the model tends to learn noise within the dataset. The most effective value, min_samples_leaf = 20, achieved the peak accuracy of 0.7820, striking a balance between generalization and model complexity by mitigating overfitting while effectively identifying significant data patterns. As min_samples_leaf was increased beyond this point (e.g., from 30 to 100), a consistent decline in accuracy was observed, indicative of underfitting, where the model's simplicity hindered its ability to capture the underlying complexities of the data. These results underscore the importance of carefully calibrating this parameter to optimize model performance Figure 3.6 –See Appendix A–.

- **Maximum Features**

In high-dimensional datasets within particle physics, it is crucial to effectively manage the complexity of ML models to avoid overfitting and minimize computational expenses. Overfitting arises when a model captures noise or irrelevant patterns in the data, resulting in reduced generalization performance on unseen data. A practical approach to address this issue involves restricting the number of features evaluated at each split in a DT, achievable through feature selection. This method not only curbs

excessive model complexity but also directs the model's focus toward the most significant features, thereby enhancing its generalization capabilities. The following parameter settings were employed: criterion function entropy, splitter best, maximum depth: 8, minimum samples for split: 50, and minimum samples per leaf: 20.

We have experimented different values of minimum samples split and wrote the accuracy of each in the Table B.1 in Appendix B.

The best accuracy obtained when the maximum features = 40.

The functions mentioned in Table B.1 –See Appendix B– pertain to the maximum features parameter in a DT, which controls the number of features considered when looking for the best split at each node. This serves as a form of feature selection, potentially reducing dimensionality and improving generalization.

- None:

When None is specified, all features (53 features) are considered for every split and may lead to overfitting. This does not reduce dimensionality and often leads to deeper, more complex trees,

- log2:

This selects a subset of features equal to the logarithm (base 2) of the total number of features, effectively reducing dimensionality by limiting the feature space. For 53 features, this would be approximately 5 or 6. However, in this dataset, reducing the number of features drastically (log2) might eliminate critical HLFs needed for jet classification, leading to lower accuracy (0.7550).

- sqrt:

Similar to log2, this function considers the square root of the total number of features for splitting. For 53 features, this would be approximately 7 or 8. It reduces dimensionality but less aggressively than log2. Still, the accuracy (0.7568) suggests that even this level of reduction may oversimplify the feature space for this dataset.

- Int (40):

Setting a fixed number of features (e.g., 40) ensures that a meaningful subset of features is consistently used, balancing dimensionality reduction and performance. In this dataset, specifying 40 features yields the highest accuracy (**0.7850**), likely because it retains enough information from the HLFs while avoiding overfitting by using fewer features per split Figure 3.7 –See Appendix A–.

As previously mentioned, there are several common metrics used to evaluate the accuracy of classification models. We utilized multiple metrics to assess the performance of the model and summarized the results in the Table B.2 –See Appendix B–.

- **Comparison Results**

1. Accuracy: 0.7850
2. Precision: 0.7859
3. Recall: 0.7850
4. F1-Score: 0.7850

This table provides a concise overview of the model's performance, highlighting its strengths in accurately classifying the jets based on the evaluated metrics. The high values across these metrics indicate the model's effectiveness in distinguishing between different types of jets. Where class 0 refer to light quarks (q), class 1 refers to gluons (g), class 2 refers to (w) bosons, class 3 refers to (z) bosons, and class 4 refers to top quarks (t).

By leveraging multiple evaluation metrics, we gain a more holistic understanding of the model's performance, ensuring that it performs well not just in terms of overall accuracy, but also in precision, recall, and the balance between them.

Depending on the previous results, in our project the best values for the hyperparameters are:

criterion function: entropy, splitter: best, maximum depth: 8, minimum samples split: 50 and minimum samples leaf: 20 and maximum features = 40.

The Table B.3 –See Appendix B– displays the recorded time (in seconds) necessary for conducting the training and evaluation phases of the DT model. The time measurements were acquired utilizing Python functions such as ‘time.time()’ to accurately capture the duration of each phase.

The results indicate that the DT model performs efficiently, with a training time of 1.1368 seconds and an evaluation time of 0.0018 seconds. The model achieves a training accuracy of 80.84% and a testing accuracy of 78.50%, suggesting a well-balanced fit with no significant overfitting or underfitting. The close alignment between training and testing accuracy highlights the model's good generalization to unseen data.

3.3 Random Forest

Similar to the DT, we used the same strategy to obtain the best hyperparameter values. We began with the default parameter set illustrated in Table B.4 –See Appendix B–, then proceeded to modify only the first one, and continued in a similar manner.

We have set `random_state=42` in `train_test_split` to ensure that the same 80% of the data is always used for training and the same 20% for testing, and we set `random_state=42` in `RandomForestClassifier` to ensure the same tree structure is built each time, this providing repeatable and clear results as shown in Figure 3.8 –See Appendix A–.

The RF with default parameters gave an accuracy of 0.8138. To enhance its performance, we will optimize it by tuning the hyperparameters.

When working with RF models for classification tasks such as classifying jets of particles in HEP, several hyperparameters are commonly tuned to optimize model performance. Here are some hyperparameters specific to RF that are commonly tuned for this task:

- **Number of Trees**

Determines the number of DT in the forest. Increasing the number of trees can improve model robustness but may increase computation time. We executed the code 20 times using different datasets and wrote the accuracy of each in the Table B.5 –See Appendix B–.

While increasing the number of trees in a RF initially improves accuracy, there is a point where the accuracy gains become negligible. Therefore, in practical applications, it is essential to balance the desired accuracy with computational efficiency.

The results from 20 experiments with varying numbers of trees in a RF model show that accuracy improves as the number of trees increases from 10 to 100, but beyond 100 trees, the accuracy gains become marginal. The performance plateaus at approximately 0.8138 accuracy, with minimal fluctuations in subsequent values even as the number of trees increases to 150, 200, or more. This highlights a trade-off between accuracy and computational cost, as increasing the number of trees beyond 100 adds computational overhead without significant improvement in performance. Therefore, 100 trees can be considered the optimal value, offering near-maximum accuracy while maintaining efficiency. The results suggest that the model's performance is stable across the experiments, and further improvements are more likely to come from fine-tuning other hyperparameters, such as maximum depth or minimum samples split, rather than increasing the number of trees Figure 3.9 –See Appendix A–.

- **Criterion Function**

Similar to the DT, we have tried different criterion functions and wrote the accuracy of each of them in the Table B.6 –See Appendix B–. We used number of trees = 80, which gave me the best accuracy.

The analysis of average accuracy across 20 experiments for different criterion functions Gini, entropy, and log_loss shows that entropy performs the best, with the highest mean accuracy of 0.8175 compared to 0.8138 for Gini and 0.8130 for log_loss Figure 3.10 –See Appendix A–.

Entropy also demonstrates greater consistency across individual experiments, with smaller variability and consistently high results. This superior performance can be attributed to entropy's focus on maximizing information gain, which leads to more balanced splits and improved generalization, especially in datasets with complex relationships or uneven class distributions. We calculated the average and variance of the results as shown in Table B.7 –See Appendix B–.

Here, the average accuracy for Entropy (0.81755) is higher than for Gini (0.81380), which suggests that Entropy achieves better accuracy on average.

We analyzed the results using One-Way ANOVA to determine whether there were any statistically significant differences between these groups with respect to our chosen variables as shown in Table B.8 –See Appendix B–.

The ANOVA results reveal a statistically significant difference between the groups, with the P-value ($P = 0.0061$) being well below the significance threshold ($\alpha = 0.05$). This indicates that the observed difference in mean accuracies between the two groups is unlikely to be due to random chance. The between-group sum of squares (SS = 0.000141) and mean square (MS = 0.000141) are considerably smaller than the within-group values (SS = 0.000634, MS = 0.000017), suggesting that most of the variability arises from differences within each group rather than between them.

The F-statistic ($F = 8.4333$) exceeds the critical value ($F_{crit} = 4.0982$), further confirming that the mean accuracies of the two groups are significantly different. This indicates that the observed improvement with Entropy is unlikely to be due to random chance. Therefore, Entropy is the superior criterion for achieving higher classification accuracy in this multi-class jet classification problem.

- **Maximum Depth**

Specifies the maximum depth of each DT in the forest. Controlling tree depth helps prevent overfitting. The accuracy results for each value of maximum depth are documented in the Table B.9 –See Appendix B–. We used the same values of hyperparameters: number of trees = 100 and criterion function: entropy.

The best accuracy obtained when the maximum depth = 10. Deeper trees do not provide significant accuracy gains and may lead to overfitting Figure 3.11 –See Appendix A–.

We compared a DT and a RF with the ensemble size = 100, to see how they differ in performance as in Figure 3.12 –See Appendix A–.

So, let's show here why we chose the maximum depth for DT =8, and for RF =10, As it is in Figure 3.12 –See Appendix A–. It can be observed in the plot that the accuracy for the train data keeps getting higher while the accuracy for the test data starts

decreasing when the max depth is larger than 8 in DT. The DT classifier overfits the training data at higher depths, leading to a decline in test accuracy. We observe that the testing accuracy peaks at a depth of 8 for the DT, making it the optimal choice. At this depth, the model achieves a balance between training and testing accuracy, avoiding overfitting while capturing sufficient complexity in the data.

On the other hand, the RF classifier, with its ensemble of 100 trees, shows greater robustness to overfitting. Its testing accuracy increases steadily up to a depth of 10 and stabilizes beyond this point. This indicates that at depth 10, the RF model strikes a balance between learning meaningful patterns and maintaining computational efficiency. Choosing a depth of 10 ensures high generalization performance while avoiding unnecessary complexity and computational overhead. Thus, the optimal maximum depths 8 for the DT and 10 for the RF are based on their respective performance trends in achieving a balance between training accuracy, test accuracy, and overfitting mitigation.

- **Minimum Samples Split**

We used GridSearchCV method as DT to adjust both maximum depth and minimum samples split in the RF. We have tested all combinations of minimum samples split values between (5-20) and maximum depth values between (5-20), and documented the results in the Table B.10 –See Appendix B–. We used the same values of variables: number of trees = 100 and criterion function: entropy. The results from the 20 datasets.

The analysis shows that the highest mean test score, 0.8325, is achieved with a maximum depth of 15 and a minimum samples split of 5. On average, configurations with a maximum depth of 10 or 20 and a minimum samples split of 5 or 10 consistently perform well. Specifically, the combination of maximum depth 10 and minimum samples split 5 yields the highest average performance across datasets (0.827825), followed closely by configurations with maximum depth 20 and minimum samples split value of 10 (0.827433).

We studied the case by using Heatmap graph to visualize the relationship between the mean test score, min_samples_split, and max_depth as shown in Figure 3.13 –See Appendix A–.

The analysis of the provided data reveals that the best performance in terms of mean test score occurs when the max_depth is set to 10 and min_samples_split is 5, yielding the highest score of 0.827825. In general, smaller values for max_depth (particularly 10) tend to result in higher mean test scores, while increasing max_depth to 15 or 20 slightly reduces performance. Additionally, a min_samples_split of 5 tends to produce better results compared to 10, suggesting that fewer samples required for splitting leads to improved model performance. Overall, the optimal configuration for this dataset is a tree with a depth of 10 and minimum samples split of 5, as it balances complexity and generalization most effectively.

- **Minimum Samples Leaf**

The "minimum samples per leaf" parameter in a RF classifier controls the minimum number of samples required to be at a leaf node. This parameter plays a critical role in determining the granularity of the splits and the overall structure of the trees within the forest. We used the same values of variables: number of trees = 100, criterion function: entropy, the maximum depth = 10 and minimum samples split = 5. We have experimented different values of minimum samples leaf and wrote the accuracy of each in the Table B.11 –See Appendix B–.

The results in Table B.11 –See Appendix B–, the mean accuracy over 20 experiments, show that the best accuracy obtained when the minimum samples leaf = 4.

Smaller values like 2 yielded a mean accuracy of 0.8260, but this may lead to overfitting as the model captures noise in the data. The optimal value, min_samples_leaf =4, provided the highest accuracy of 0.7820, balancing generalization and complexity by avoiding overfitting while capturing key patterns in the data. As min_samples_leaf increased further (e.g., 10 to 50), the accuracy steadily declined due to underfitting, where the model became too simplistic to capture the data's intricacies. These findings demonstrate that tuning this parameter is critical for achieving optimal performance Figure 3.14 –See Appendix A–.

Depending on the previous results, in our experiment the best values for the hyperparameters are:

Number of trees = 100, criterion function: entropy, the maximum depth = 10 and minimum samples split = 5 and minimum samples leaf = 4.

The Table B.12 –See Appendix B– presents the measured time (in seconds) required to execute the training and evaluation processes for RF model.

The RF model demonstrates strong training performance with an accuracy of 90.55%, but shows signs of overfitting as evidenced by a lower testing accuracy of 82.95%. The model's training time of 12.3123 seconds is relatively long, reflecting its complexity, while the evaluation time of 0.0469 seconds remains efficient for practical use. To improve generalization and reduce overfitting, we used feature importance (feature selection) method and recorded the results as shown in the Table B.12 –See Appendix B–.

When training time are reduced, it indicates that feature selection has made the model more efficient. As shown the percentage of reduction between training time with feature selection and training time without feature selection are 40%. This is particularly useful when dealing with large datasets or when computational resources are limited. But we have slightly decrease in training accuracy, accompanied by a stable testing accuracy, suggests that feature selection has helped in reducing overfitting. But it may affect accuracy, due to the abandonment of some features. The model now has a more balanced performance between the training and testing datasets.

In summary, the RF model reached a reasonable accuracy quickly, by applying feature importances for feature selection, the RF model has become more efficient, with a notable decrease in training time and a slight reduction in overfitting, as indicated by the closer alignment of training and testing accuracies. The model's performance on unseen data remains robust, demonstrating that the most important features have been successfully identified and utilized. This approach helps streamline the model, making it more practical for deployment while maintaining high predictive accuracy.

We repeated our experiments 20 times without fixing the random seed for DT and RF, we got the following results as shown in Table B.13 –See Appendix B–. Where class 0 refer to light quarks (q), class 1 refers to gluons (g), class 2 refers to (w) bosons, class 3 refers to (z) bosons, and class 4 refers to top quarks (t).

The analysis of Table B.13–See Appendix B– highlights that the RF model consistently outperforms the DT model across all classes in precision, recall, F1-score, and overall accuracy (85.32% vs. 81.32%). RF is particularly effective in identifying W bosons, Z bosons, and top quarks, but both models show lower performance for light quarks (Class 0) and gluons (Class 1). The reason is probably that it is often these particle types often produce jets with very similar physical characteristics, leading to significant overlap in their feature space. As a result, even well-performing models may struggle to distinguish between them.

The possible sources of error, one reason might be that the features used don't clearly separate some of the classes. These results suggest using RF as the preferred model and focusing on improving feature representation and addressing class imbalances to enhance classification for weaker-performing classes. Exploring alternative ensemble methods like Gradient Boosting could further boost performance.

In a jet classification task in high energy physics, both DT and RF demonstrate their respective strengths and weaknesses. When we fixed random seed the DT model, with a training time of 1.1368 seconds and evaluation time of 0.0018 seconds, achieved a training accuracy of 80.84% and a testing accuracy of 78.50%. This suggests a quick and straightforward model, though it may not fully capture the complexity of the data. In contrast, the RF model, after applying feature importance selection, had a longer training time of 7.7873 seconds and an evaluation time of 0.0469 seconds, but it achieved higher training accuracy of 88.45% and testing accuracy of 82.60%. But when we allowed randomness (no fixed random seed) improves performance DT achieved consistent and reasonable accuracy (81.32%) and F1-scores (ranging from 0.66 to 0.88 across classes), and RF achieved accuracy = 85.32%.

- DT provide a reliable method for jet classification, particularly when simplicity and interpretability are key priorities. However, their performance is limited compared to RF, which consistently achieve higher accuracy and better generalization. While RF models are more computationally intensive, they offer superior performance by combining multiple DT, reducing overfitting, and improving robustness. This makes RF a more effective solution for complex classification tasks in HEP, despite their greater computational demands.

- The computational complexity and resource requirements for DT and RF models are crucial considerations in HEP, where large datasets are common. Given the hardware constraints (an Intel i5 processor and 8GB RAM), both models can be executed effectively, but RF push the limits of resource utilization due to their higher computational requirements. For extremely large datasets common in HEP, leveraging more powerful hardware or distributed computing resources may be necessary for training RF models efficiently.
- We used number of trees = 100, criterion function: entropy, the maximum depth = 10 and minimum samples split = 5 and minimum samples leaf = 4 for RF and, criterion function: entropy, splitter: best, maximum depth: 8, minimum samples split: 50 and minimum samples leaf: 20 and maximum features = 40 for DT, until we reached the highest possible accuracy.
- RF with feature selection is more efficient as it reduces training time by approximately 36.8% (from 12.31 seconds to 7.79 seconds) while maintaining comparable accuracy. DT is faster to train (1.1368 seconds) but underperforms in accuracy and may not be the best choice for jet classification where high accuracy is critical.
- From the feature ranking results, the top 20 features contribute the most to RFs performance. This indicates that these features are sufficient for achieving high accuracy without using all 53 HLF features.
- Based on RFs feature ranking, the top 20 significant features are:

1. Feature 7 (`b'j_tau3_b1'`): N-subjettiness ($\tau 3$) with $\beta=1$, indicating the likelihood of a jet having a three-prong structure.
2. Feature 50 (`b'j_mass_prun'`): Pruned jet mass, a jet mass after pruning to remove soft, wide-angle radiation.
3. Feature 47 (`b'j_n2_b2_mmdt'`): N2 variable ($\beta=2$) calculated after the Modified Mass Drop Tagger (mMDT) grooming algorithm.
4. Feature 15 (`b'j_c1_b1'`): First-order energy correlation function with $\beta=1$.
5. Feature 3 (`b'j_eta'`): Pseudorapidity of the jet, describing its angular position relative to the beamline.
6. Feature 29 (`b'j_tau3_b1_mmdt'`): N-subjettiness ($\tau 3$) with $\beta=1$, calculated after mMDT grooming.
7. Feature 19 (`b'j_d2_b1'`): D2 variable with $\beta=1$, a ratio of energy correlation

functions used for discrimination between jet types.

8. Feature 1 (`b'j_ptfrac'`): Fractional transverse momentum of the jet.
9. Feature 36 (`b'j_c1_b1_mmdt'`): First-order energy correlation function with β=1, calculated after mMDT grooming.
10. Feature 48 (`b'j_mass_trim'`): Trimmed jet mass, a jet mass calculated after trimming soft radiation.
11. Feature 17 (`b'j_c2_b1'`): Second-order energy correlation function with β=1.
12. Feature 14 (`b'j_c1_b0'`): First-order energy correlation function with β=0.
13. Feature 52 (`b'j_mass_sdm1'`): Soft Drop mass with β=-1.
14. Feature 49 (`b'j_mass_mmdt'`): Jet mass calculated after the Modified Mass Drop Tagger (mMDT) grooming algorithm.
15. Feature 35 (`b'j_c1_b0_mmdt'`): First-order energy correlation function with β=0, calculated after mMDT grooming.
16. Feature 9 (`b'j_tau2_b2'`): N-subjettiness (τ2) with β=2.
17. Feature 25 (`b'j_n2_b1'`): N2 variable with β=1, used for jet type discrimination.
18. Feature 6 (`b'j_tau2_b1'`): N-subjettiness (τ2) with β=1.
19. Feature 24 (`b'j_m2_b2'`): M2 variable with β=2, a higher-order energy correlation function.
20. Feature 8 (`b'j_tau1_b2'`): N-subjettiness (τ1) with β=2, indicating a one-prong jet structure.

These features should be prioritized for model training or feature engineering as they contribute the most to classification accuracy.

- The best way to construct DT or RF to classify jets:

i. DT: Use max_features= 40 as a hyperparameter. its performance (accuracy: 81.32%). Optimal parameters include (Max depth=8, splitter function= "best", minimum sample split=50).
ii. RF: Use feature selection (top 20 features) to balance efficiency and accuracy 85.32%. Feature selection improves both interpretability and training efficiency without compromising performance significantly. Optimal parameters include (Max depth=10, Number of estimators=100, minimum sample split=5).

This study compares the use of DT and RF with a thesis that employs NN for jet classification in HEP. The NN approach demonstrates strong capabilities in modeling intricate, non-linear relationships, achieving an accuracy of 74.00% through sophisticated architectural designs. In contrast, our DT and RF models emphasize interpretability, computational efficiency, and the evaluation of feature importance, with the RF model attaining a superior accuracy of 85.32%. Both methodologies align with the overarching goal of utilizing artificial intelligence to enhance jet classification, underscoring the inherent trade-offs between model complexity and predictive performance. Together, these approaches deepen our insights into particle interactions within complex environments.

Comparing the results of this research with the JEDI-net model, which relies on advanced interactive networks and achieves an accuracy of up to 93.00% using extensive computational resources and precise parameter tuning, it is clear that using simpler models such as random forests (RF) and decision trees (DT) can deliver remarkable performance despite technical limitations. In a test environment based on an Intel i5-5200U processor and 8GB of RAM, the RF model achieved a classification accuracy of 85.32%. Although JEDI-net employs complex deep learning techniques and a large number of parameters, this research demonstrates that adopting feature selection strategies and careful parameter tuning significantly improves efficiency and reduces training time, making these models an effective choice when advanced computing resources are unavailable.

Chapter Four

Conclusion and Future Work

In this section, we provide a comprehensive overview of our study and outline potential avenues for future research, drawing on the significant insights and advancements achieved through the application of DT and RF methodologies in the domain of particle classification.

1.4 Conclusion

This study focuses on the advancement and optimization of DT and RF methodologies to enhance precision in particle classification, addressing various particle categories commonly encountered in HEP research. By conducting comprehensive experiments and systematic parameter adjustments, significant enhancements in classification performance were attained. The primary outcomes and key contributions of this work are outlined below:

- DT and RF Model Design: We used DT and RF methodologies to construct a classification system for identifying particle types within jets, with the aim of classifying jets into five categories: light quarks (q), gluons (g), W and Z bosons, and top quarks.
- DT provide a reliable method for jet classification but, RF achieve superior accuracy (85.32%) and generalization compared to DT (81.32%).
- Optimal parameters for DT: criterion function: entropy, splitter: best, maximum depth: 8, minimum samples split: 50, minimum samples leaf: 20 and maximum features = 40.
- Optimal parameters for RF: number of trees = 100, criterion function: entropy, the maximum depth = 10 and minimum samples split = 5, minimum samples leaf = 4 and top 20 features for feature selection.
- RF with feature selection is more efficient as it reduces training time by 36.8% while maintaining high accuracy.
- The top 20 features are sufficient for high RF performance, improving efficiency and interpretability.
- Based on RF feature ranking, the top 20 significant features are: [N-subjettiness ($\tau 3$) with $\beta=1$, Pruned jet mass, N2 variable ($\beta=2$) after mMDT, First-order energy

correlation function (β=1), Jet pseudorapidity, N-subjettiness (τ3) with β=1, after mMDT, D2 variable with β=1, Fractional transverse momentum, First-order energy correlation (β=1), after mMDT, Trimmed jet mass, Second-order energy correlation function (β=1), First-order energy correlation function (β=0), Soft Drop mass (β=-1), Jet mass after mMDT, First-order energy correlation (β=0), after mMDT, N-subjettiness (τ2) with β=2, N2 variable with β=1, N-subjettiness (τ2) with β=1, M2 variable with β=2, N-subjettiness (τ1) with β=2].

- The best way to construct DT or RF to classify jets: DT: use max_features= 40 as a hyperparameter. its performance (accuracy: 81.32%) and, RF: use feature selection (top 20 features) to balance efficiency and accuracy 85.32%. Feature selection improves both interpretability and training efficiency without compromising performance significantly.
- Research Implications: Our findings provide valuable guidance for researchers tackling similar pattern recognition challenges across disciplines. They highlight the advantages of ensemble methods like RF for complex classification tasks and emphasize the importance of detailed parameter tuning for high-performance results.

This study shows that using traditional machine learning methods like Random Forest (RF) and Decision Tree (DT) can be very effective in classifying particle jets in high energy physics (HEP). Although these models are simpler than deep learning approaches, the RF model achieved a high accuracy of 85.32% when optimized and supported by feature selection. Importantly, this was done using only modest computing resources. These results suggest that simple and easy-to-understand models can still perform well, making them a practical option in physics experiments where computing power may be limited.

2.4 Future Work

Future research endeavors will concentrate on several key areas to advance the field of particle classification:

- Incorporation of Diverse Data Sources: Subsequent investigations should explore the integration of supplementary data sources, including advanced sensor data, to enhance the reliability and robustness of DT and RF models in particle classification tasks.

- Development of Hybrid Models: Research should delve into the development of hybrid models that merge RF with other ML methodologies, such as Gradient Boosting Machines (GBM) or NN architectures. Such hybrid approaches may yield significant improvements in classification accuracy and performance.
- Refinement of Feature Engineering and Selection: Further efforts should be directed toward refining feature engineering and selection processes to identify the most pertinent features for particle classification. Employing techniques like recursive feature elimination (RFE) or principal component analysis (PCA) could contribute to enhanced model accuracy and efficiency.

In summary, this study underscores the efficacy of optimized DT and RF algorithms in particle classification. By pursuing the outlined research directions, we can advance particle identification methodologies and gain deeper insights into the fundamental particles that constitute the universe.

List of Abbreviations

Abbreviation	Meaning
DT	Decision Tree
RF	Random Forest
HLFs	High Level Features
LHC	Large Hadron Collider
ML	Machine Learning
HEP	High Energy Physics
NN	Neural Network

References

[1] Albertsson, K., Altoe, P., Anderson, D., Andrews, M., Araque Espinosa, J. P., Aurisano, A.,... & Zapata, O. (2018). Machine learning in high energy physics community white paper. *Journal of Physics: Conference Series, 1085*(1), 022008. https://doi.org/10.1088/1742-6596/1085/2/022008

[2] Alnuaimi, A. F., & Albaldawi, T. H. (2024). An overview of machine learning classification techniques. *BIO Web of Conferences, 97*, 00133. https://doi.org/10.1051/bioconf/20249700133

[3] Alsagheer, R. H., Alharan, A. F., & Al-Haboobi, A. S. (2017). Popular decision tree algorithms of data mining techniques: A review. *International Journal of Computer Science and Mobile Computing, 6*(6), 133-142.

[4] AlSagri, H., & Ykhlef, M. (2020). Quantifying feature importance for detecting depression using random forest. *International Journal of Advanced Computer Science and Applications, 11*(5), 1-10. https://doi.org/10.14569/IJACSA.2020.0110505

[5] Alzubi, J., Nayyar, A., & Kumar, A. (2018). Machine learning from theory to algorithms: An overview. *Journal of Physics: Conference Series, 1142*(1), 012012. https://doi.org/10.1088/1742-6596/1142/1/012012

[6] Ashari, A., Paryudi, I., & Tjoa, A. M. (2013). Performance comparison between Naïve Bayes, decision tree and k-nearest neighbor in searching alternative design in an energy simulation tool. *International Journal of Advanced Computer Science and Applications, 4*(11), 33-38. https://doi.org/10.14569/IJACSA.2013.041115

[7] Bhasin, H. (2020). *Machine learning for beginners: Learn to build machine learning systems using Python*. BPB Publications.

[8] Boussaada, A. (2019). *Tunisian truck license plate recognition using an Android application based on machine learning as a detection tool* [Master's thesis, Hochschule Merseburg]. Hochschulbibliothek.

[9] Boyko, N., Omeliukh, R., & Duliaba, N. (2022). The random forest algorithm as an element of statistical learning for disease prediction. *Interactions, 4*(1), 13-24.

[10] Müller, A. C. (2017). *Introduction to machine learning with Python*. Springer.

[11] Costa, V. G., & Pedreira, C. E. (2023). Recent advances in decision trees: An updated survey. *Artificial Intelligence Review, 56*(5), 4765-4800. https://doi.org/10.1007/s10462-022-10294-2

[12] De Castro Manzano, P. (2019). *Statistical learning and inference at particle collider experiments* (CERN-THESIS-2019-209) [Doctoral dissertation, Padua University]. CERN. https://cds.cern.ch/record/2701341

[13] Dedov, F. (2020). *The Python bible 7 in 1: Volumes one to seven (Beginner, Intermediate, Data Science, Machine Learning, Finance, Neural Networks, Computer Vision)*. Independently published.

[14] Duarte, J., Han, S., Harris, P., Jindariani, S., Kreinar, E., Kreis, B.,... & Wu, Z. (2018). Fast inference of deep neural networks in FPGAs for particle physics. *Journal of Instrumentation, 13*(7), P07027. https://doi.org/10.1088/1748-0221/13/07/P07027

[15] Fletcher, S., & Islam, M. Z. (2019). Decision tree classification with differential privacy: A survey. *ACM Computing Surveys, 52*(4), 1-33. https://doi.org/10.1145/3337064

[16] Friligkos, G., Papaioannou, E., & Kaklamanis, C. (2023). A framework for applying the logistic regression model to obtain predictive analytics for tennis matches. *Journal of Sports Analytics, 9*(2), 45-60.

[17] Fu, Y., Yin, Z., Su, M., Wu, Y., & Liu, G. (2020). Construction and reasoning approach of belief rule-base for classification based on decision tree. *IEEE Access, 8*, 138046-138057. https://doi.org/10.1109/ACCESS.2020.3012163

[18] Gates, M. (2017). *Machine learning for beginners: Definitive guide for neural networks, algorithms, random forests and decision trees made simple*. CreateSpace Independent Publishing Platform.

[19] Géron, A. (2022). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow* (2nd ed.). O'Reilly Media.

[20] Ghiasi, M. M., & Zendehboudi, S. (2021). Application of decision tree-based ensemble learning in the classification of breast cancer. *Computers in Biology and Medicine, 128*, 104089. https://doi.org/10.1016/j.compbiomed.2020.104089

[21] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

[22] Gupta, R. (2020). A survey on machine learning approaches and its techniques. *2020 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS)*, 1-6. https://doi.org/10.1109/SCEECS48394.2020.114

[23] Hamoud, A., Hashim, A. S., & Awadh, W. A. (2018). Predicting student performance in higher education institutions using decision tree analysis. *International Journal of Interactive Multimedia and Artificial Intelligence, 5*(2), 26-31. https://doi.org/10.9781/ijimai.2018.02.004

[24] Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). Springer.

[25] James, G., Witten, D., Hastie, T., Tibshirani, R., & Taylor, J. (2023). *An introduction to statistical learning: With applications in Python*. Springer. https://doi.org/10.1007/978-3-031-38747-0

[26] Kan, X., & Li, T. (2020). Slant split criterion random forests classification algorithm based on soft margin hyperplane. *2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, *9*, 1332-1337. https://doi.org/10.1109/ITAIC49862.2020.9339042

[27] Rabbertz, K. (2018). *Jet physics at the LHC: The strong force beyond the TeV scale*. Springer.

[28] Kozak, J. (2019). *Decision tree and ensemble learning based on ant colony optimization* [Unpublished doctoral dissertation]. University of Warsaw.

[29] Lee, C. S., Cheang, P. Y. S., & Moslehpour, M. (2022). Predictive analytics in business analytics: Decision tree. *Advances in Decision Sciences, 26*(1), 1-29. https://doi.org/10.47654/v26y2022i1p1-29

[30] Lefevre, C. (2008). *LHC: The guide* (CERN-Brochure-2008-001-Eng). CERN.

[31] Luo, H., Luo, M. X., Wang, K., Xu, T., & Zhu, G. (2019). Quark jet versus gluon jet: Fully-connected neural networks with high-level features. *Science China Physics, Mechanics & Astronomy, 62*(1), 1-11. https://doi.org/10.1007/s11433-018-9364-0

[32] Mahesh, B. (2020). Machine learning algorithms—A review. *International Journal of Science and Research, 9*(1), 381-386.

[33] Maimon, O. Z., & Rokach, L. (2014). *Data mining with decision trees: Theory and applications* (2nd ed.). World Scientific.

[34] Maimon, O., & Rokach, L. (Eds.). (2005). *Data mining and knowledge discovery handbook* (2nd ed.). Springer.

[35] Marzani, S., Soyez, G., & Spannowsky, M. (2019). *Looking inside jets: An introduction to jet substructure and boosted-object phenomenology* (Vol. 958). Springer. https://doi.org/10.1007/978-3-030-15709-8

[36] Mehta, P., Bukov, M., Wang, C. H., Day, A. G., Richardson, C., Fisher, C. K., & Schwab, D. J. (2019). A high-bias, low-variance introduction to machine learning for physicists. *Physics Reports, 810*, 1-124. https://doi.org/10.1016/j.physrep.2019.03.001

[37] Miah, M. O., Khan, S. S., Shatabda, S., & Farid, D. M. (2019). Improving detection accuracy for imbalanced network intrusion classification using cluster-based under-sampling with random forests. *2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)*, 1-5. https://doi.org/10.1109/ICASERT.2019.8934543

[38] Mienye, I. D., Sun, Y., & Wang, Z. (2019). Prediction performance of improved decision tree-based algorithms: A review. *Procedia Manufacturing, 35*, 698-703. https://doi.org/10.1016/j.promfg.2019.07.005

[39] Moreno, E. A., Cerri, O., Duarte, J. M., Newman, H. B., Nguyen, T. Q., Periwal, A.,... & Vlimant, J. R. (2020). JEDI-net: A jet identification algorithm based on

interaction networks. *The European Physical Journal C, 80*(1), 1-15. https://doi.org/10.1140/epjc/s10052-020-7607-5

[40] Noshad, Z., Javaid, N., Saba, T., Wadud, Z., Saleem, M. Q., Alzahrani, M. E., & Sheta, O. E. (2019). Fault detection in wireless sensor networks through the random forest classifier. *Sensors, 19*(7), 1568. https://doi.org/10.3390/s19071568

[41] Patel, H. H., & Prajapati, P. (2018). Study and analysis of decision tree based classification algorithms. *International Journal of Computer Sciences and Engineering, 6*(10), 74-78.

[42] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O.,... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825-2830.

[43] Rao, C. R., & Gudivada, V. N. (2018). *Computational analysis and understanding of natural languages: Principles, methods and applications*. Elsevier.

[44] Salih, A. A., & Abdulazeez, A. M. (2021). Evaluation of classification algorithms for intrusion detection system: A review. *Journal of Soft Computing and Data Mining, 2*(1), 31-40. https://doi.org/10.30880/jscdm.2021.02.01.004

[45] Sen, P. C., Hajra, M., & Ghosh, M. (2020). Supervised classification algorithms in machine learning: A survey and review. In *Emerging technology in modelling and graphics* (pp. 99-111). Springer. https://doi.org/10.1007/978-981-13-7403-6_11

[46] Shamrat, F. J. M., Chakraborty, S., Billah, M. M., Das, P., Muna, J. N., & Ranjan, R. (2021). A comprehensive study on pre-pruning and post-pruning methods of decision tree classification algorithm. *2021 5th International Conference on Trends in Electronics and Informatics (ICOEI)*, 1339-1345. https://doi.org/10.1109/ICOEI51242.2021.9452957

[47] Sharma, M., Sharma, H., Bhutani, P., & Sharma, I. (2021). Credit card fraud detection using machine learning algorithms. In *Innovations in cyber physical systems* (pp. 547-560). Springer. https://doi.org/10.1007/978-981-15-8295-5_48

[48] Sharma, S., & Sharma, S. K. (2020). A study on machine learning tools. *IITM Journal of Management and IT, 11*(1), 98-102.

[49] Sheykhmousa, M., Mahdianpari, M., Ghanbari, H., Mohammadimanesh, F., Ghamisi, P., & Homayouni, S. (2020). Support vector machine versus random forest for remote sensing image classification: A meta-analysis and systematic review. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 13*, 6308-6325. https://doi.org/10.1109/JSTARS.2020.3026724

[50] Singh, N., & Singh, P. (2019). A novel Bagged Naïve Bayes-Decision Tree approach for multi-class classification problems. *Journal of Intelligent & Fuzzy Systems, 36*(3), 2261-2271. https://doi.org/10.3233/JIFS-169932

[51] Subudhi, A., Dash, M., & Sabut, S. (2020). Automated segmentation and classification of brain stroke using expectation-maximization and random forest

classifier. *Biocybernetics and Biomedical Engineering, 40*(1), 277-289. https://doi.org/10.1016/j.bbe.2019.12.003

[52] Sullivan, W. (2017). *Machine learning beginners guide algorithms: Supervised & unsupervised learning, decision tree & random forest introduction*. CreateSpace Independent Publishing Platform.

[53] Suthaharan, S. (2016). *Machine learning models and algorithms for big data classification*. Springer. https://doi.org/10.1007/978-1-4899-7641-3

[54] Tangirala, S. (2020). Evaluating the impact of GINI index and information gain on classification using decision tree classifier algorithm. *International Journal of Advanced Computer Science and Applications, 11*(2), 612-619. https://doi.org/10.14569/IJACSA.2020.0110277

[55] Trzciński, T., Graczykowski, Ł., Glinka, M., & ALICE Collaboration. (2020). Using random forest classifier for particle identification in the ALICE experiment. In *Information technology, systems research, and computational physics 3* (pp. 3-17). Springer. https://doi.org/10.1007/978-3-030-43987-2_1

[56] Zhou, C., Yu, H., Ding, Y., Guo, F., & Gong, X. (2017). Decision tree for classifying Betacoronavirus species using amino acid frequencies. *PLOS ONE, 12*(8), e0181426. https://doi.org/10.1371/journal.pone.0181426

[57] Zhou, Y., Cheng, G., Jiang, S., & Dai, M. (2020). Building an efficient intrusion detection system based on feature selection and ensemble classifier. *Computer Networks, 174*, 107247. https://doi.org/10.1016/j.comnet.2020.107247

Appendices

Appendix A

Figures

Figure A.1

Chart Varying Maximum Depth of DT

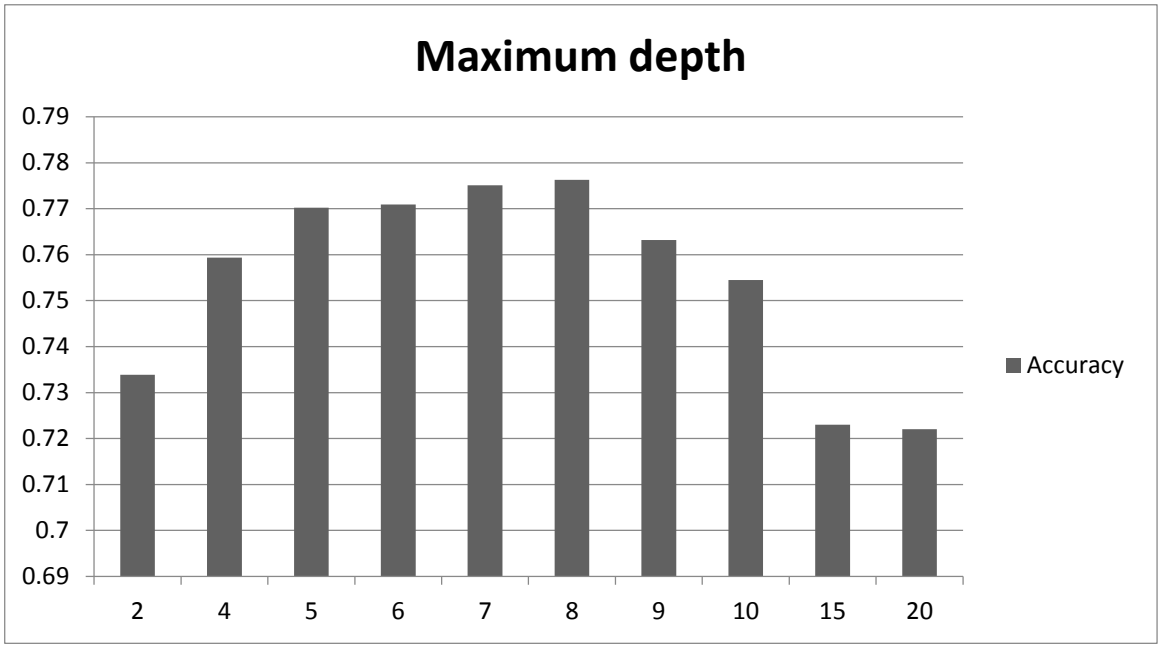


Figure A.2

Heatmap of Mean Test Scores for Max Depth and Min Samples Split for DT

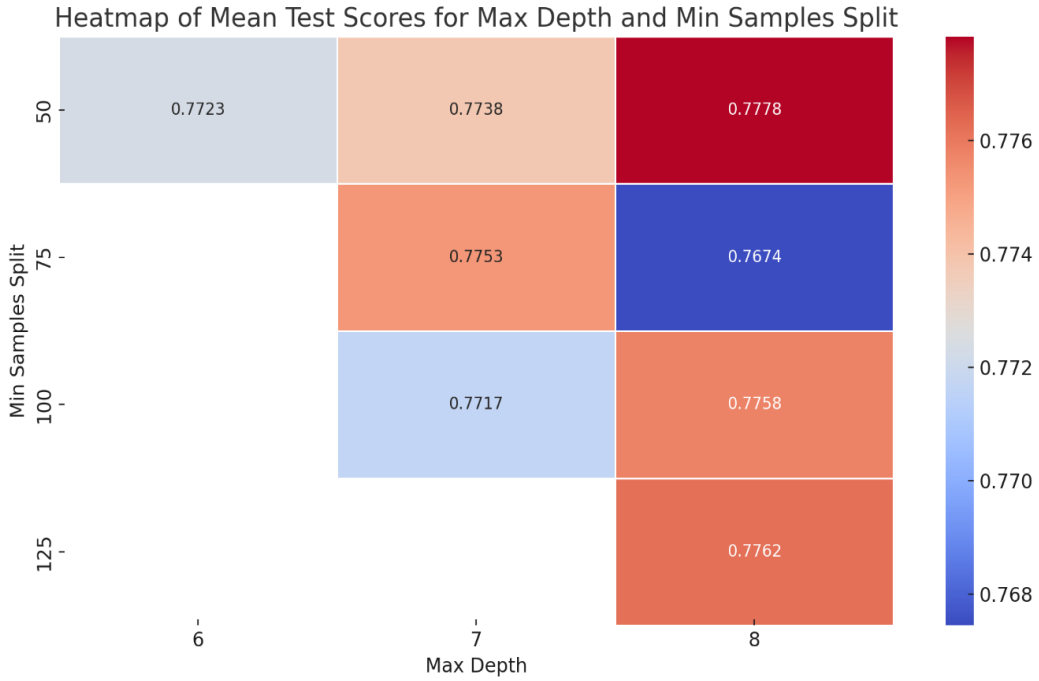


Figure A.3

Chart Varying Minimum Samples Leaf of DT

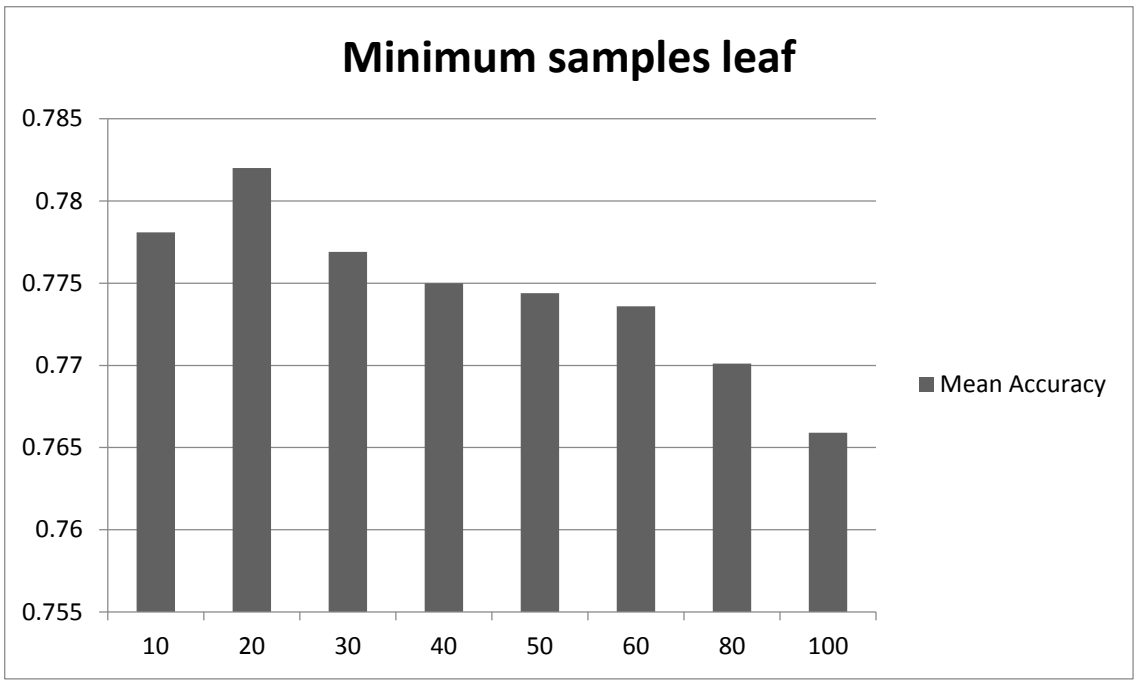


Figure A.4

Chart Varying Maximum Features of DT

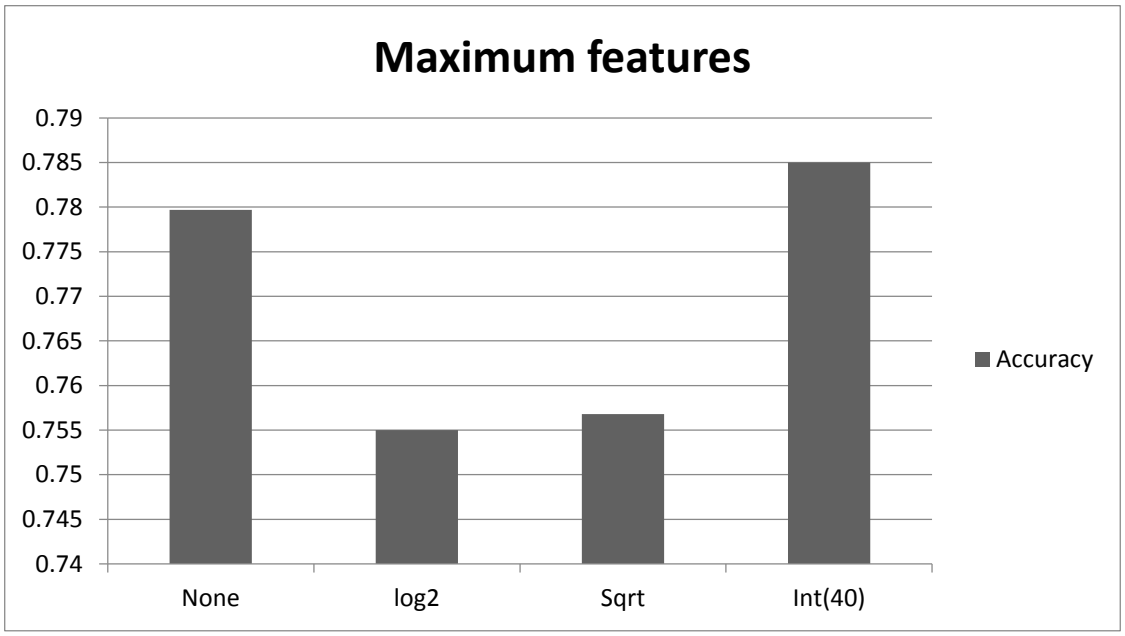


Figure A.5

Random Forest with Default Parameters Code

```
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
x_train,x_test,y_train,y_test= train_test_split(X,Y,test_size=0.20,random_state=42)
clf = RandomForestClassifier(random_state=42)
clf.fit(x_train,y_train)
y_pred=clf.predict(x_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Figure A.6

Chart Varying Number of Trees of RF

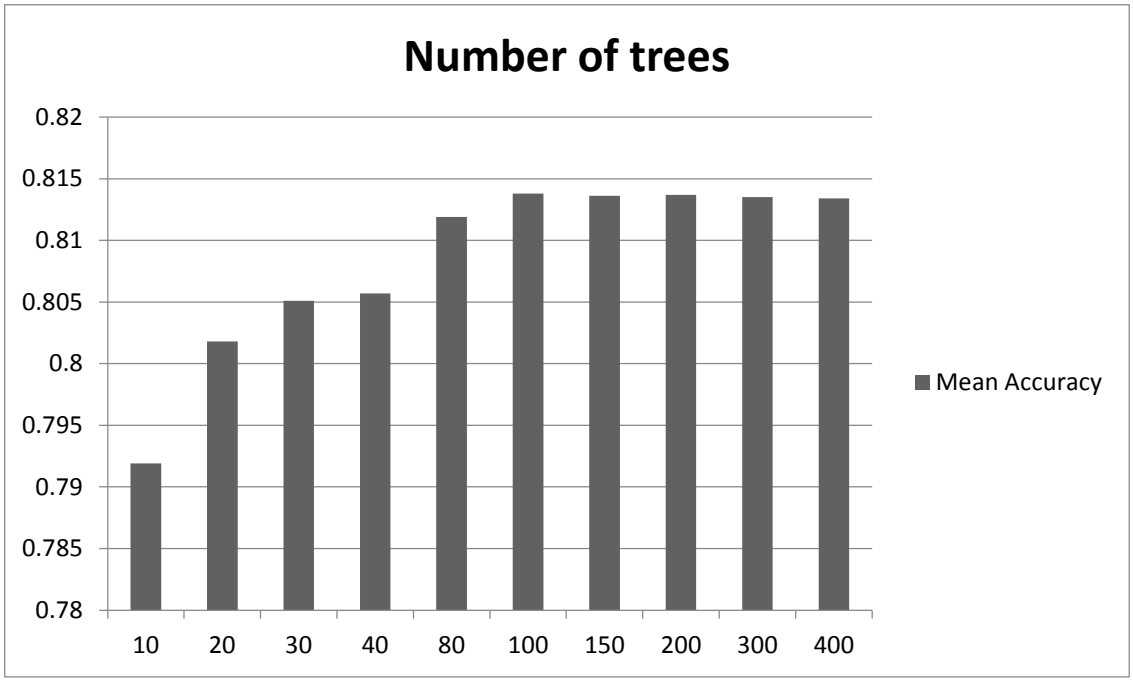


Figure A.7

Chart Varying Criterion Function of RF

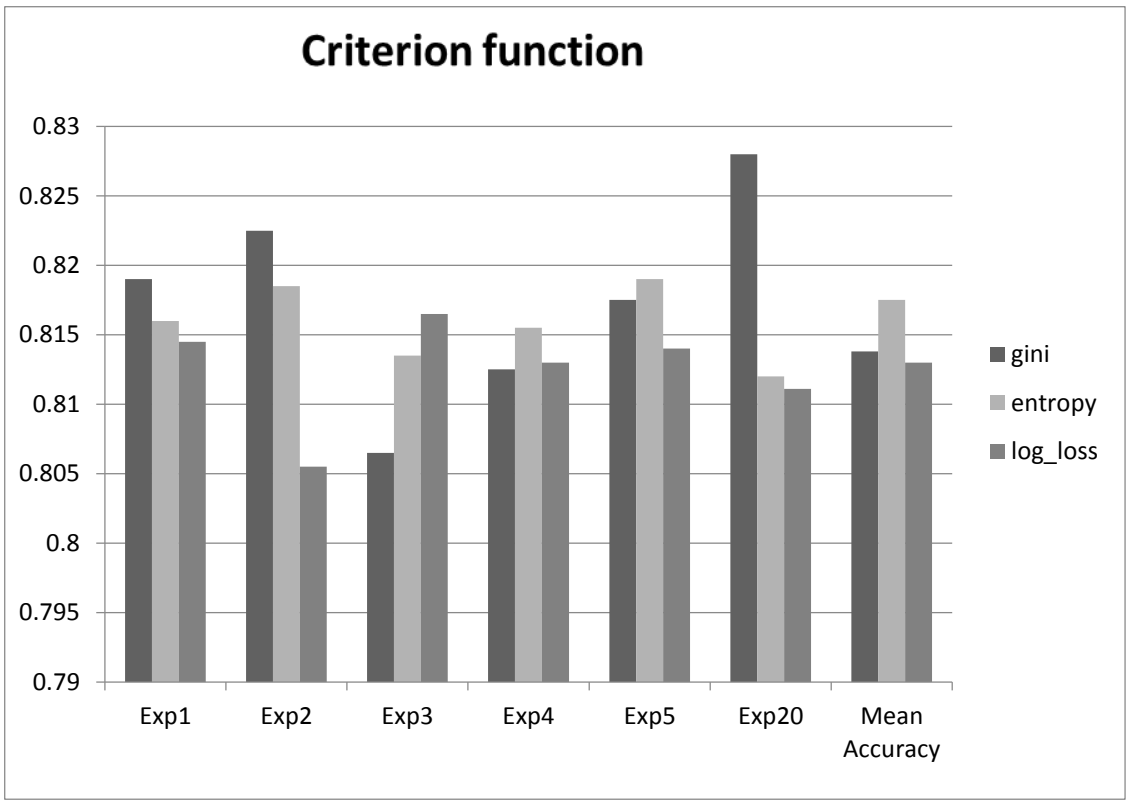


Figure A.8

Chart Varying Maximum Depth of RF

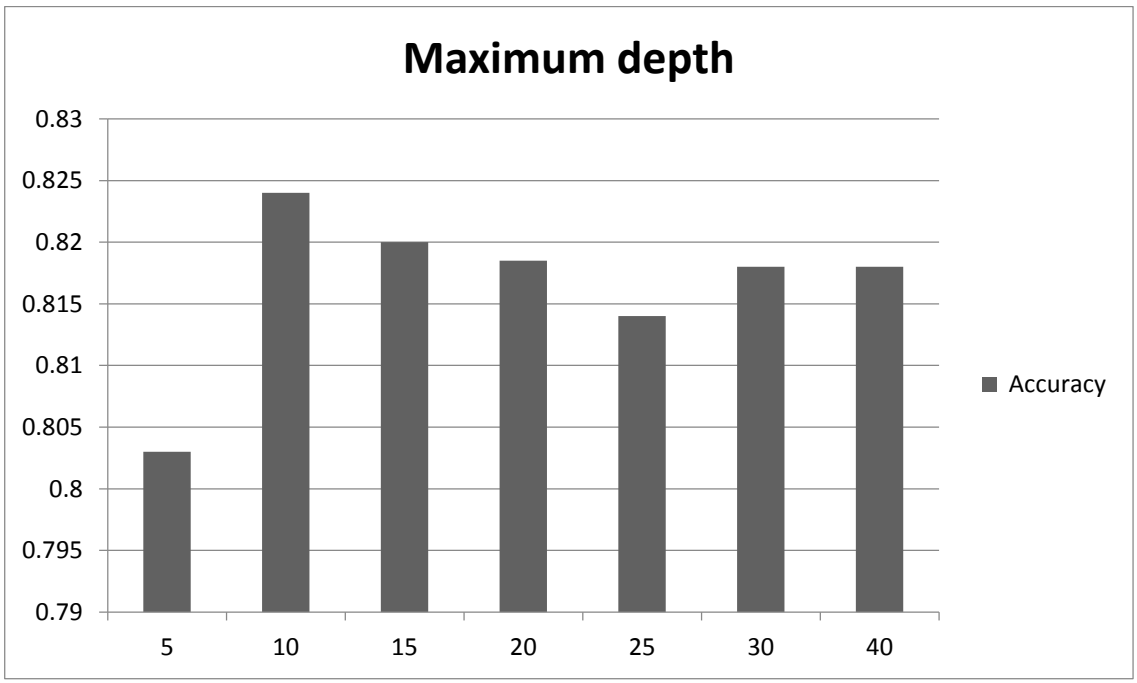


Figure A.9

Maximum Depth for DT Vs RF with assembling size =100

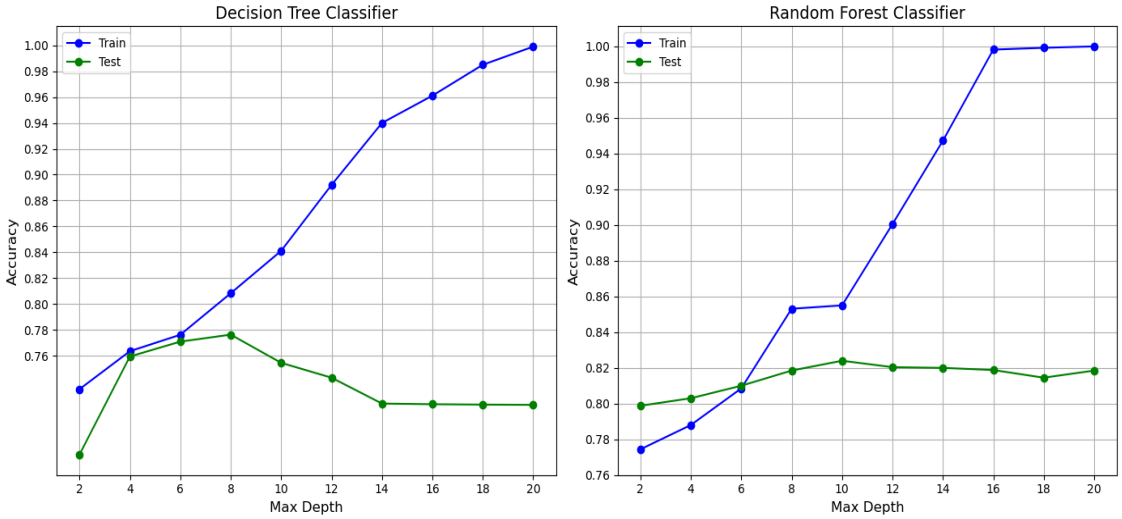


Figure A.10

Heatmap of Mean Test Scores for Max Depth and Min Samples Split for RF

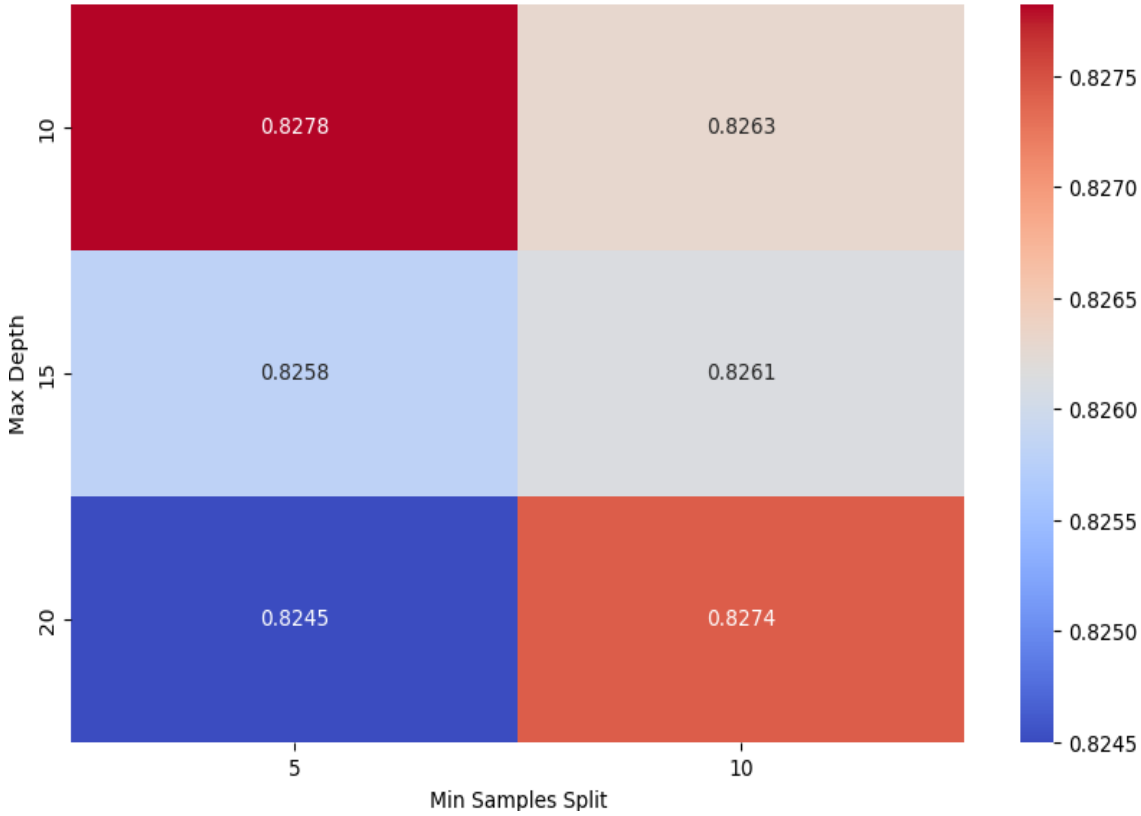
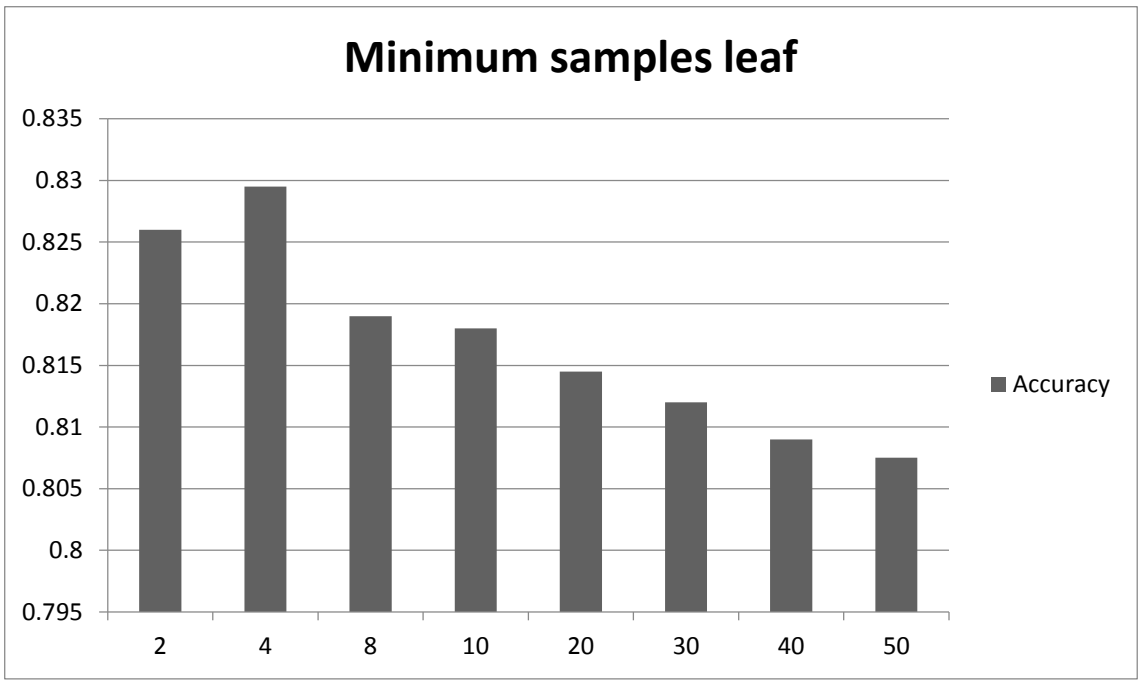


Figure A.11

Chart Varying Minimum Samples leaf of RF



Appendix B

Tables

Table B.1

Accuracy of Different Number of Maximum Features of DT

Maximum features	Accuracy
None (53 features)	0.7797
log2 (5 or 6 features)	0.7550
Sqrt (7 or 8 features)	0.7568
Int (40 features)	0.7850

Table B.2

Classification Report for DT After Taking Average for 20 Experiments

Class	Precision	Recall	F1-Score	Support
0	0.73	0.75	0.74	428
1	0.70	0.67	0.69	389
2	0.86	0.82	0.84	356
3	0.87	0.85	0.86	402
4	0.78	0.84	0.81	425

Table B.3

Training and Evaluation Time of DT Model

Training Time	Evaluation Time	Training Accuracy	Testing Accuracy
1.1368 seconds	0.0018 seconds	0.8084	0.7850

Table B.4

A Summary of the Hyperparameters Used for RF

Hyperparameter	Description
Number of Trees	The number of DT in the forest. Higher numbers may improve accuracy but increase computational cost (default: 100).
Criterion Function	The function to measure the quality of a split (default: Gini).
Maximum Depth	The maximum depth of each tree. Limits how deep each tree can grow (default: None, meaning the tree grows until all leaves are pure).
Minimum Samples Split	The minimum number of samples required to split an internal node (default: 2).
Minimum Samples Leaf	The minimum number of samples required to be at a leaf node (default: 1).

Table B.5

Accuracy of Different Number of Trees for RF

Number of trees	Mean Accuracy
10	0.7919
20	0.8018
30	0.8051
40	0.8057
80	0.8119
100	0.8138
150	0.8136
200	0.8137
300	0.8135
400	0.8134

Table B.6

Accuracy of Different Criterion Functions of RF

Criterion function	Exp1	Exp2	Exp3	Exp4	Exp5	Exp20	Mean Accuracy
gini	0.8190	0.8225	0.8065	0.8125	0.8175	0.8280	0.8138
entropy	0.8160	0.8185	0.8135	0.8155	0.8190	0.8120	0.8175
log_loss	0.8145	0.8055	0.8165	0.8130	0.8140	0.8111	0.8130

Table B.7

Summary for Two Groups (Entropy vs. Gini) RF

SUMMARY				
Groups	*Count*	*Sum*	*Average*	*Variance*
Entropy	20	16.351	0.81755	0.000022
Gini	20	16.276	0.81380	0.000011

Table B.8

Anova: Single Factor for Two Groups (Entropy vs. Gini) RF

ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	0.000141	1	0.000141	8.433283	0.006107	4.098172
Within Groups	0.000634	38	0.000017			
Total	0.000774	39				

Table B.9

Accuracy of Varying Maximum Depth of RF

Maximum depth	Mean Test Accuracy
5	0.8030
10	0.8240
15	0.8200
20	0.8185
25	0.8140
30	0.8180
40	0.8180

Table B.10

Accuracy of Varying Minimum Samples Split and Maximum Depth of RF

Dataset number	Minimum samples split	Maximum depth	Mean test score
1	10	20	0.8240
2	5	15	0.8325
3	10	15	0.8320
4	5	20	0.8225
5	10	15	0.8273
6	5	20	0.8291
7	5	20	0.8210
8	10	15	0.8285
9	10	20	0.8267
10	10	15	0.8236
11	10	15	0.8200
12	5	10	0.8288
13	10	10	0.8263
14	10	20	0.8316
15	5	15	0.8196
16	5	10	0.8293
17	5	10	0.8286
18	5	20	0.8254
19	10	15	0.8249
20	5	10	0.8246

Table B.11

Accuracy of Varying Minimum Samples Leaf of RF

Minimum samples leaf	Mean Accuracy
2	0.8260
4	0.8295
8	0.8190
10	0.8180
20	0.8145
30	0.8120
40	0.8090
50	0.8075

Table B.12

Training and Evaluation Time and Accuracy of RF Model

Situation	Training Time (in seconds)	Evaluation Time (in seconds)	Training Accuracy	Testing Accuracy
Without using Feature Selection	12.3123	0.0469	0.9055	0.8295
With Using Feature Selection	7.7873	0.0469	0.8845	0.8260

Table B.13

Average Classification Report for DT and RF Without Fixing Random Seed After Taking Average for 20 Experiments:

Class	Decision Tree Results			Random Forest Results		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
0	0.69	0.63	0.66	0.76	0.72	0.74
1	0.72	0.65	0.68	0.78	0.75	0.76
2	0.85	0.77	0.81	0.86	0.84	0.85
3	0.84	0.79	0.81	0.89	0.88	0.88
4	0.81	0.83	0.82	0.91	0.86	0.88
Accuracy		0.8132			0.8532	

جامعة النجاح الوطنية
كلية الدراسات العليا

شجرة القرار لتصنيف نفاثات الجسيمات في فيزياء الطاقة العالية

إعداد
مروة عبد الفتاح خالد

إشراف
د. بكر عبدالحق

قدمت هذه الرسالة استكمالا لمتطلبات الحصول على درجة الماجستير في الحوسبة المتقدمة،
من كلية الدراسات العليا، في جامعة النجاح الوطنية، نابلس – فلسطين.

2025

شجرة القرار لتصنيف نفاثات الجسيمات في فيزياء الطاقة العالية

إعداد

مروة عبد الفتاح خالد

إشراف

د. بكر عبدالحق

الملخص

لقد أصبحت تقنيات تعلم الآلة عنصراً جوهرياً في البحث العلمي، حيث تقدم حلولاً فعّالة لمشاكل علمية وهندسية متنوعة. من بين الأساليب الحديثة التي أثبتت كفاءتها في التصنيف والتحليل، تبرز "شجرة القرار" و"الغابة العشوائية" كأدوات قوية في حل المسائل المعقدة والمتعددة الأبعاد. تهدف هذه الدراسة إلى الاستفادة من شجرة القرار والغابة العشوائية لبناء نظام تصنيف فعّال للقياسات التي يتم إجراؤها في مسرعات الجسيمات، بهدف تصنيف "jets" إلى خمس فئات أساسية: الكواركات الخفيفة "light quarks " ، الجلونات " gluons " ، بوزونات "W" و"Z"، والكواركات العلوية " top quarks ".

تركز الدراسة على تصميم واختبار نماذج شجرة القرار والغابة العشوائية، مع تحسين المعاملات المؤثرة في دقة التصنيف مثل عمق الشجرة، وعدد العينات الدنيا لكل فرع، وعدد الأشجار في الغابة. يتم إجراء تجارب دقيقة لضبط هذه المعاملات بهدف تحقيق أفضل مستويات دقة التصنيف، مع الحفاظ على الكفاءة الحسابية.

تشير التوقعات إلى أن نظام تصنيف "jets" المستند إلى شجرة القرار والغابة العشوائية سيكون قادرًا على تحقيق دقة عالية وكفاءة حسابية ممتازة، مما يعزز من إمكانات التطبيق العملي لهذه النماذج في مجال فيزياء الجسيمات. تسهم هذه الدراسة في تطوير طرق متقدمة في تعلم الآلة، وتفتح آفاقاً لتطبيقات متنوعة في الأبحاث العلمية والهندسية.

الكلمات المفتاحية: تعلم الآلة، شجرة القرار، الغابة العشوائية، تحسين المعاملات، تصنيف الجسيمات، المسرعات الذرية.