ØSTFOLD UNIVERSITY COLLEGE

MASTER'S THESIS

ANOMALY DETECTION & NOVEL DATA PREPROCESSING FRAMEWORK FOR UNSW-NB15 NETWORK INTRUSION DATASET

JUNE 2022

A Novel Data Preprocessing Framework & Comprehensive Performance Evaluation for the UNSW-NB15 Network Intrusion Dataset

Authors

Sajepan Gnanasivam
Daniel Tveter

Department of Computer Science and Communications

Faculty of Computer Science, Engineering and Economy

Abstract

Cyber-attacks are becoming more complex, resulting in increased difficulty detecting anomalies accurately. In light of this, Anomaly Detection is also a rapidly evolving field of study in Computer Science and Machine Learning (ML) to counteract this exact problem. The original Anomaly Detection method required human interactions to detect abnormal values within a system or dataset. As a result, many Intrusion Detection datasets have been created, both artificially and by data collection, so that researchers have a space to implement advanced Machine Learning methods and techniques to mitigate cyber-attacks. However, the effectiveness of these techniques decreases caused by high-dimensional data.

Although there is a fair amount of research regarding Anomaly Detection using Intrusion Detection datasets, the results have considerable limitations. For example, the Machine Learning algorithm the research focuses on, performs considerably better than other algorithms they compare. In addition, they mainly direct their preprocessing techniques toward a specific algorithm, making it challenging to compare other research results.

Therefore, our thesis aims to implement techniques for Supervised Anomaly Detection with Binary Classification. Furthermore, we explore crucial aspects of data preprocessing and analysis by proposing our Novel Data Preprocessing Framework (NDPF), combined with hyperparameter tuning for existing Supervised Machine Learning algorithms for the UNSW-NB15 dataset. Our proposed NDPF implements feature selection and data transformation techniques critical to maximizing results without affecting the classification process.

Furthermore, the thesis consists of quantitative data exploration and documentation methods. In addition, our research presents an extensive performance analysis for our implemented algorithms using statistical computation. Six popular ML algorithms: Artificial Neural Network (ANN), Support Vector Machines (SVM), K-Nearest Neighbor (KNN), Logistic Regression (LR), Decision Trees (DT), and Random Forest (RF), have been implemented with Binary Classification. In addition, we determine the performance of the algorithms using performance metrics such as Accuracy, Recall, Precision, and F1 Score. Further, we hyperparameter tune manually and with the help of the Grid Search algorithm to get the best possible parameters for each algorithm, maximizing our chosen performance metrics.

The results indicate that our NDPF performs well-balanced across the six chosen Machine Learning algorithms. Furthermore, we concluded from the results of our

experimentation that the weakest performing algorithm is LR, whereas the most robust algorithm proved to be ANN. The percentage difference in performance between these two algorithms is documented to be 6.46% for training accuracy and 8.29% for testing accuracy. Compared to other research, our NDPF outperforms by an overall balance in results. Hence, our study in the thesis is highly recommendable for other research.

Keywords— Machine Learning, Anomaly Detection, Supervised Machine Learning algorithm, Data Preprocessing Framework, Hyperparameter Tuning

Abbrevations

AD	Anomaly Detection
AF	Activation Function
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
BCE	Binary Cross-Entropy
C4.5	Cervical Segment 4/5
CART	Classification and Regression Trees
CSV	Comma Separated Variable
DDoS	Distributed Denial of Service
DoS	Denial of Service
DT	Decision Trees
EDA	Exploratory Data Analysis
ELU	Exponential Linear Unit
FNN	Feedforward Neural Network
GAN	Generative Adversarial Networks
GD	Gradient Descent
GELU	Gaussian Error Linear Unit
GS	Grid Search
ID3	Iterative Dichotomiser 3
IDS	Intrusion Detection System
IG	Information Gain
KDE	Kernel Density Estimation

KNN	K-Nearest Neighbor
LR	Logistic Regression
MAE	Mean Absolute Error
ME	Mean Error
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
NDPF	Novel Data Preprocessing Framework
NN	Neural Network
PReLU	Parametric ReLU
RBF	Radial Basis Function
ReLU	Rectified Linear Unit
RF	Random Forest
RFE	Recursive Feature Elimination
RFECV	Recursive Feature Elimination Cross Validation
RMSE	Root Mean Squared Error
RS	Random Search
RSS	Residual Sum of Squares
SELU	Scaled Exponential Unit
SLP	Single Layer Perceptron
SNN	Self-normalizing Neural Network
SOP	Sum of Products
SVM	Support Vector Machines

Acknowledgments

First and foremost, we are incredibly grateful to our supervisor Nga Dinh, for her continuous support, invaluable guidance, and patience during our M.Sc study. Her substantial expertise and knowledge have aided us throughout this academic year. Our gratitude extends to the Faculty of Computer Science, Engineering and Economy for the opportunity to undertake our studies at the Department of Computer Science and Communications at Østfold University College. Finally, we would like to express our gratitude to our family and friends for their tremendous patience and encouragement.

Prerequisites

It is advantageous for the reader to have prior knowledge of Machine Learning, Python Programming, Statistics, Linear Algebra, and Calculus. Every subject with technical and theoretical aspects will be explained and presented in appropriate sections. We have structured our thesis so that the majority of readers get a friendly introduction to the technology we utilize.

Contents

List of Figures

List of Tables

List of Code

Chapter 1

Introduction

The massive growth in cyber threats, coupled with modern organizations' reliance on the stability and effectiveness of their IT infrastructure, has prompted a shift in mentality. As a result, priorities are altering as the downtime increases for systems within information technology. Furthermore, intrusions, such as brute force, Denial of Service (DoS), Worms, Exploits, Backdoors, Reconnaissance, Shellcode, or even penetration within a network, are the most common threat to a network's security. Similarly, cybercriminals are violently attempting to interrupt network connections, get illegal access to essential information, and then steal, distort, or damage essential data structures. Hence, a dynamic approach to detecting and preventing such intrusions is required. It is a critical challenge in the field of computer network security that must be addressed. In light of the substantial number of sophisticated attacks, anomaly detection has been hugely popular[1].

1.1 Anomaly Detection

Network anomaly detection aims to find network abnormalities, where supervised classification approaches have yielded several successes through many types of research[2]–[4]. The difficulty of discovering unusual patterns in network traffic that do not conform to predicted typical behavior is called anomaly-based intrusion detection in networks. In many applications, these nonconforming patterns are referred to as anomalies, aberrations, outliers, or surprises[5]. Efforts have been undertaken over the last three decades to develop automated systems for detecting network irregularities[6]. Anomaly detection in data networks is a vital and rapidly evolving field of study since our every day is becoming more and more data-driven. With more data and information available than ever before, it is critical to analyze and evaluate it properly [7]. In a data network, anomaly detection has a wide range of applications, including credit card fraud detection, cyber security intrusion detection, and military observation of adversary activity. An unusual traffic pattern in a computer network, for example, could indicate that a hacked computer is transferring sensitive data to an unauthorized site[5]. According to Cisco's Annual Internet Report, the frequency of breaches and the number of records exposed to each breach increase. Between 2018 and 2019, attacks between 100 Gbps and 400 Gbps

increased by 776%, and the total number of DDoS attacks will double from 7.9 million in 2018 to 15.4 million by 2023[8].

Traditionally, Anomaly Detection was carried out manually by human supervisors. Its drawbacks are discussed in a conference paper by Bram Steenwinckel[9], where the author quotes:

> *"Anomaly detection (AD) systems are either manually built by experts setting thresholds on data or constructed automatically by learning from the available data through machine learning (ML)."*

Building an Anomaly Detection system is demanding since implementation requires domain knowledge of the application and skillful implementation capabilities. However, the help of ML and its ability to predict, makes ML a prime choice for Anomaly Detection.

1.2 Machine Learning

Machine Learning[10] is a field of Artificial Intelligence (AI) and Computer Science, as shown in Figure 1.1. ML aims to replicate human behavior in the sense of repeating and gradually increasing precision and accuracy in a task. The term ML was introduced in a paper by Arthur L. Samuel in 1959[11] regarding the game checkers, whereas a professional human checkers player was beaten by the computer (IBM 7094). This exact moment was founded to be a significant milestone for Artificial Intelligence.

ML is a crucial part of the rapidly expanding domain of Data Science[1]. Various ML algorithms are trained to output classification or regression predictions utilizing statistical approaches, revealing crucial data insight. Following that, these findings support decision-making within software applications and enterprises to add growth for their assets. Generally, as big data expands and grows, the demand for Data scientists will rise, necessitating their assistance in identifying the most critical business questions and, as a result, the data needed to answer them[12].

1.2.1 Anomaly Detection with Machine Learning

Machine learning for anomaly detection necessitates a thorough grasp of the topic, mainly when handling unstructured data. On the other hand, structured data provides data with information regarding the area of the subject. Moreover, it is easier for a specific model to detect when an abnormal event occurs since it defies specific regulations. However, unstructured data, such as media files (images, mp3, recordings, and videos), carries minimal meaning and renders the algorithms ineffective until the data is organized (structured). Secondly, to develop a good model, we need extensive data collection. Any successful ML model needs datasets as a foundational premise. There has to be good quality data to back up our claim for our model's predictions. Especially, large data

[1]The discipline of obtaining information or insight from diverse sorts of data is known as data science. Data Science is a field of study that integrates mathematics, statistics, and informatics.

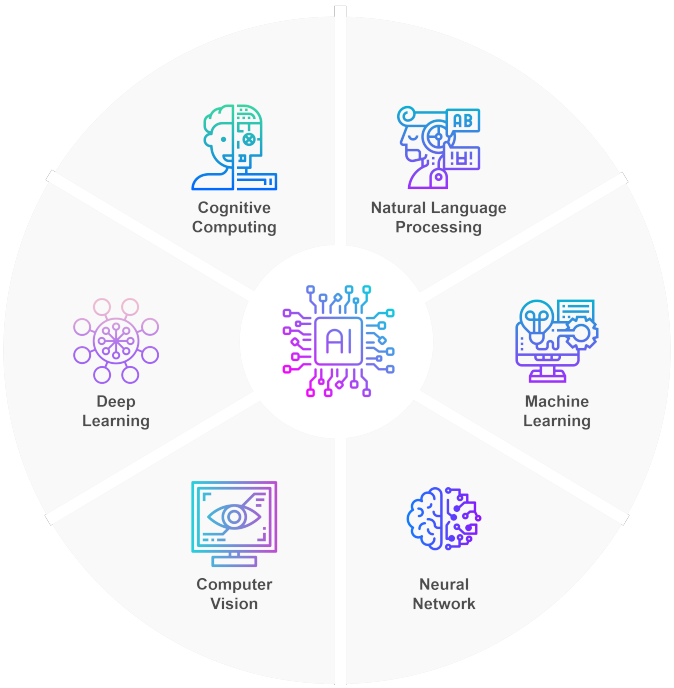


Figure 1.1: Branches of Artificial Intelligence

collection is beneficial for anomaly detection since a specific model can conclude that an anomaly is infrequent compared to *normal*[2] occurrences.

Many anomaly detection techniques, mainly *Unsupervised Learning* (Chapter 3, Section 3.3.2), have an issue where a significant change (anomaly) in values is hard to detect since the data is unlabeled. However, Supervised Learning (Chapter 3, Section 3.3.1) approaches needs a labeled dataset, with a feature marking each row of data as ”normal” or ”anomaly”[13]. In light of this, our thesis has mainly focused on Supervised Learning approaches for anomaly detection, resulting in the need for a structured and labeled dataset.

1.3 Dataset Selection

What are the characteristics of a good intrusion dataset? It might be challenging to get a reliable dataset for anomaly detection studies caused by the very nature of technology. Furthermore, technology is a domain that evolves quickly, where the threats also evolve with them. As a result, datasets lose their relevance within an insignificant amount of time. Hence, researchers have developed a set of requirements for prevalent datasets[14]. One of the crucial properties of intrusion detection datasets is the variety of attack

[2]In the field of Anomaly Detection with ML, the term *normal* refers to the occurrences that are not an abnormal value.

categories that tries to cover all of the possible threats. Businesses invest significant sums of money in implementing security countermeasures for adversaries who threaten their systems. As a result, adversaries adapt and evolve their attack methods to penetrate said systems, so using datasets created with updated attack categories is essential. Computer networks can be phenomenally huge and complicated; hence, the dataset must be created in a realistic environment. Labeling datasets is very crucial, although it can be difficult. Labels that reflect the real-world attribute are necessary for calculating our models' accuracy, whether supervised- or unsupervised learning. Datasets must also be well-received by the scientific community. When researching anomaly detection, our findings are more likely to be approved by the community if the dataset is relatable[15].

In light of this, we have researched the capabilities of several datasets presented in Table 1.1, to determine which one to utilize for our thesis.

Table 1.1: Network Intrusion Datasets

Dataset			
1	KDD-Cup 1999	**4**	CICIDS 2017
2	Gure-KDD	**5**	Bot-IoT
3	NSL-KDD	**6**	UNSW–NB15

1.3.1 Dataset Review

The first dataset we took into consideration was KDD-Cup 1999[3][16]. The creators of the dataset used a cyber range, which allowed cyber-security specialist to exercise network assaults against real-world targets. A cyber range consists of actual network equipment. The data from the Cyber range was captured and analyzed by Bro-IDS, an Intrusion Detection System. As a result, the data was acquired in the form of several CSV files, where the project researchers processed and labeled the datasets. The KDD-Cup 1999 dataset consists of 4.900.000 rows of data organized into four attack categories: DoS, user-to-root, remote-to-local, and probing. The whole dataset, including the attack categories and normal data, is labeled, making it ideal for supervised- and unsupervised learning. Since it was the sole dataset that permitted research to be extended across various researchers, KDD-Cup 1999 has been the gold standard for intrusion detection datasets. However, as time went by, the community started to criticize the realism of the dataset. First, a dataset from 1999 had less relevancy for research twenty years later since technology and its cyber security measure evolved and changed. Secondly, they criticized the dataset further for the redundant data (duplicate data). For example, data will be duplicated in the cases of DoS attacks, which is expected, but the community voted down regardless since duplicate data can lead to *overfitting*[4]. Finally, the community criticized the size of the dataset. Researchers only used a subset of the dataset for training and

[3] `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`

[4] Overfitting is explained in Chapter 3, Section 3.2.1

testing, and when different projects select different subsets of the same dataset, it makes it challenging to analyze and compare results.

To address this criticism, variations of the KDD-Cup 1999 dataset were developed. For example, Gure KDD[5] was a variation of the KDD-Cup 1999, where they addressed the issue of detecting the different cyber-attacks. In addition, the researchers added the RAW network, which included the payload and header data. Another dataset, NSL-KDD[6], approached the criticism for the KDD-Cup 1999 by addressing the distribution of the dataset by removing duplicate and unnecessary data. Unfortunately, both variations of the KDD-Cup 1999, the Gure KDD and NSL-KDD shared the same underlying flaw: they were built on outdated technology that was no longer relevant[15].

The UNSW-NB15[17], is a recent dataset that meets several of the requirements for a robust dataset. The dataset was developed in 2015, utilizing a network traffic generator to output artificial network data, including regular and simulated attack data. Researchers used the Bro-IDS to handle the raw network data, similar to how the KDD-Cup 1999 dataset was generated. They also employed Argus, an extra security tool, to add additional features. As a result, the researchers developed nine distinct forms of attacks and in addition to normal data. Further, UNSW-NB15 has been acknowledged by the research community, resulting in appearances in several peer-reviewed journals[18]–[20]. Furthermore, newer intrusion detection datasets have been developed named *CICIDS2017*[7] and *BoT-IoT*[8]. The CICIDS2017 and BoT-IoT datasets are not chosen since UNSW-NB15 is more well-established and appreciated by the community. Hence, we tend to choose to use the UNSW-NB15 for our thesis for the reasons mentioned. In addition, there is a substantial amount of research done on it already, making it easier for us to compare the results in Chapter 6, Section 6.8[15].

1.4 Motivation

Anomaly Detection is not a subject that can be concluded with a definite solution since technology and its threats grow equally. Therefore, the solution for the problem has to be dynamic and updated continuously based on the threat. If the problem is not handled by developing regularly, the consequences can be immense in the worst case. To put the problem in perspective, i.e., a massive cyber-attack on a medical sector can result in a system shutdown, where the consequences can be fatal.

There are many studies on Anomaly Detection, where the primary limitation lies in the dataset's preprocessing[21], [22], which means that research mainly focuses on data preprocessing directed to enhance the performance of one specific algorithm. For example, in research, author *Kasongo, et al.*[21] introduced a Performance Analysis of an intrusion detection system with a feature selection method. As a result, their research received the highest score of 88.13% for Decision Trees (DT) and the lowest score of 62.42% for

[5] `http://www.aldapa.eus/res/gureKddcup/README.pdf`
[6] `https://www.unb.ca/cic/datasets/nsl.html`
[7] `https://www.unb.ca/cic/datasets/ids-2017.html`
[8] `https://research.unsw.edu.au/projects/bot-iot-dataset`

SVM regarding supervised Binary Classification. On the other hand, author *Suleiman et al.*[22] conducted similar research regarding supervised classification algorithms. As a result, their research received the highest score of 90.14% for RF and the lowest score of 81.91%. Furthermore, *Kasongo, et al.* indicates that the difference between the best and worst-performing models is 41.18%, and *Suleiman et al.*, 10.04%. This results in a well-performing model with their chosen algorithm, but other popular algorithms cannot compare. This analysis is done in Chapter 6.

1.5 Problem Statement and Objectives

As we have established and will further explain, Cyber-attacks are a severe issue within the private and public sectors. Consequently, Anomaly Detection has emerged as a popular and forthcoming topic in Computer Science to mitigate this exact problem. According to our literature study (Chapter 2), there is a tremendous amount of solid research on anomaly detection, especially for our chosen dataset, UNSW-NB15.

In order to mitigate this problem, our thesis aims to develop a Novel Preprocessing Framework (NDPF) to transform the UNSW-NB15 dataset to outperform several Supervised ML algorithms, compared to other research. An extensive performance evaluation is conducted on the dataset while considering Binary Classification for six algorithms: Artificial Neural Networks (ANN), Support Vector Machines (SVM), K-Nearest Neighbor (KNN), Logistic Regression (LR), Decision Trees (DT), and Random Forest (RF), where the algorithms are explained in Chapter 4. Furthermore, we hyperparameter-tune our prediction models to maximize our performance metrics. Finally, our thesis aims to develop a novel hybrid algorithm that works well with our preprocessing framework. We have categorized our objectives into four different main categories for a better overview (Table 1.2): Background Analysis (B), Novel Data Preprocessing Framework (P), Algorithm Implementation (A), and Evaluation (E).

1. **Background Analysis (B)** aims to gather the necessary information to execute our thesis. In the Background Analysis, we have investigated which dataset to conduct our experiments on, preprocessing methods, performance results from other research, review, and selection of ML algorithms to conduct our experiments with, and finally, the development environment.

2. **Novel Data Preprocessing Framework (P)** aims to implement the techniques and information gathered on preprocessing methods from the Background Analysis (B). The goal is to create a Novel Data Preprocessing Framework which performs well across multiple ML algorithms.

3. **Algorithm Implementation (A**) firstly aims to implement our selected ML algorithms from Background Analysis (B), where we maximize our performance metrics by extensive Hyperparameter tuning. Second, we aim to develop a novel algorithm comparable to existing ones.

4. **Evaluation (E)** aims to conduct a comprehensive performance evaluation with comparable results against existing research.

Table 1.2: Thesis Objectives

Main Category	Purpose	Sub Category	Objective	Description
Background Analysis (B)	To review and determine appropriate dataset and methods for our thesis	**B1**	Dataset Selection	Select a suitable and well-balanced Network Intrusion Dataset.
		B2	Preprocessing methods	Review preprocessing methods available.
		B3	Performance results for other works	Review other works for chosen dataset.
		B4	Algorithm Review	Review popular Machine Learning algorithms.
		B5	Algorithm Selection	Select appropriate algorithms for selected dataset.
		B6	Development Environment	Select Development environment and programming language.
Novel Data Preprocessing Framework (P)	To create a well-performing dataset across multiple algorithms, by developing a Novel Data Preprocessing Framework	**P1**	***P1.1*** Data Cleaning	Remove missing and redudant data.
			P1.2 Feature Selection	Select the most optimal features.
			P1.3 Feature Engineering	Create or remove features manually based on exisitng data.
			P1.4 Data Transformation	Transform the dataset, and prepare for implementation.
		P2	***P2.1*** Feature Selection with algorithms	Automatic Feature Selection with RFE.
			P2.2 Feature Selection, own method	Random Forest Classifier with 10-fold Cross-validation.
Algorithm Implementaiton (A)	Implement our chosen algorithms, and hyperparameter tune our models to maximize our results.	**A1**	***A1.1*** Implementation of ANN	Implement the algorithms for the preprocessed Dataset. Hyperparameter tuning for maximizing results.
			A1.2 Implementation of SVM	
			A1.3 Implementation of KNN	
			A1.4 Implementation of LR	
			A1.5 Implementation of DT	
			A1.6 Implementation of RF	
	Develop a novel algorithm for our dataset that performs better or comparable than existing ones.	**A2**	Develop a Novel Algorithm	Develop a Novel algorithm, for our preprocessed dataset.
Evaluation (E)	Present evidence that our method performs well across multiple algorithms, compared to other existing works.	**E1**	***E1.1*** Comparable results for ANN	Result Comparison with other works evaluating the corresponding algorithms.
			E1.2 Comparable results for SVM	
			E1.3 Comparable results for KNN	
			E1.4 Comparable results for LR	
			E1.5 Comparable results for DT	
			E1.6 Comparable results for RF	
		E2	Visualize Results	Visualize the results in an informative way.

1.6 Contribution

The main contributions of our thesis have been developed to accomplish our objectives stated in Table 1.2. In summary, the contributions of our research are:

1. **Novel Data Preprocessing Framework:** We have developed a Novel Data Preprocessing Framework for the UNSW-NB15 dataset. The framework applies several critical techniques to output a dataset that performs accurately and balanced for multiple algorithms which is proven to be stable.

2. **Experimentations and Hyperparameter tuning:** We have conducted extensive experimentations with Hyperparameter tuning to achieve the best performance metrics possible for each model, where our supervised ML algorithms were implemented with Binary Classification in mind. In our case, the Binary Classification output will either be 0 (non-attack/normal) or 1 (attack). The hyperparameter tuning has been accomplished with Manual- and the Grid Search to select the best possible parameters for each algorithm, maximizing our results.

3. **Performance Evaluation:** as a final step, we test our NDPF in combination with hyperparameter tuning on our selected algorithms. Numerous experiments are conducted for each algorithm to support our final result. Furthermore, the results

from our experimentations are gathered for a performance analysis against each other. We determine which algorithm performed the best among the six. Finally, we conduct a comprehensive performance evaluation against other research, where our proposed framework is proven to outperform and be balanced across multiple ML algorithms, as seen in Chapter 6, Section 6.8

1.7 Thesis Structure

The rest of this thesis is organized as follows:

- **Chapter 2**, *Related Works* introduces existing research from the scientific community regarding our chosen dataset, UNSW-NB15. The literature study has been conducted to get an apprehension of efforts that have been made on our topic so we do not overlap and create duplicate research.
- **Chapter 3**, *Essential Concepts* introduces the reader to essential ML concepts such as data, parameters, hyperparameters, overfitting, underfitting, supervised- and unsupervised learning, which will be used throughout the thesis.
- **Chapter 4**, *Machine Learning Algorithms* introduces in detail the different ML algorithms used for our thesis, where the algorithms are presented both in text and visually appealing figures for better understanding. The included algorithms in the section are ANN (4.1), LR (4.2), SVM (4.3), DT (4.4), RF (4.5), and KNN (4.6).
- **Chapter 5**, *Implementation* firstly introduces the different tools and libraries (5.1) used to develop our project. Furthermore, the Novel Preprocessing Framework has been explained in detail, including the different phases: Data cleaning-, transformation, and selection(5.3). Finally, we describe how we implemented the algorithms mentioned in Chapter 4.
- **Chapter 6**, *Performance Evaluation* firstly introduces the Performance Metrics we have relied on to determine whether our NDPF performed well or not. The metrics we use are Accuracy, Recall, Precision, and F1 Score. Furthermore, the experiments and their results for our chosen algorithms are presented. Finally, a Performance Analysis Summary (6.8) compares our results against other research papers.
- **Chapter 7**, *Conclusion* summarizes the whole thesis regarding our research findings. This final chapter also includes some intriguing findings and recommendations for future work.

Chapter 2

Related Work

This Chapter presents state-of-the-art ML algorithms for Anomaly Detection. We focus on existing works with the same chosen dataset, UNSW-NB15, which is a public dataset for benchmarking purposes. In addition, this Chapter also reviews some deep learning approaches used for classification with the same dataset.

2.1 Supervised and Unsupervised ML Algorithms

Both supervised, and unsupervised ML algorithms are used for Anomaly Detection. Supervised ML algorithms assume the distinction in the dataset, which is labeled as 'normal' or 'abnormal'. The supervised ML algorithms are divided further into parametric or non-parametric. The parametric supervised ML algorithm summarizes the data with a fixed set of parameters independent of the sample size. LR and Naive Bayes are kinds of this type. On the other hand, the non-parametric supervised ML algorithm does not have any parameters. Instead, they are subject to change depending on the size of the dataset[23]. KNN, DT, and SVM belong to non-parametric supervised ML algorithms. Unlike supervised ML algorithms, the unsupervised ones are built without the dataset being labeled. Unsupervised ML algorithms are further divided into clustering, association mining, and outlier mining. The clustering model differentiates each cluster by evaluating the given training dataset. Association mining is to identify events that often occur together. On the other hand, outlier mining aims to find the pattern which is different from the majority of dataset[23]. In the scope of this thesis, we focus on Anomaly Detection detection using supervised ML algorithms with both parametric and non-parametric.

2.2 Anomaly Detection in UNSW-NB15

There has been a considerable amount of research on using ML algorithms for Anomaly Detection on the UNSW-NB15 Dataset. A performance analysis was studied in[21]. The authors dealt with both Multi-Class Classification and Binary Classification in

this research. They first implemented a feature extraction method called a filter-based feature reduction technique using the extreme gradient boosting (XGBoost) algorithm to prune some features which do not impact the classification process. The authors then implemented several ML algorithms, including SVM, KNN, ANN, and DT. Their results demonstrated that the proposed XGBoost-based feature selection method improved most algorithms' Test Accuracy in Binary Classification. Significantly, in the case of the DT algorithm, the Test Accuracy was improved from 88.13% to 90.85%. In addition, the Test Accuracy was slightly improved in the case of Multi-Class Classification. Unlike their works, we only focus on Binary Classification due to time constraints. However, we test our proposed framework for most of the existing classical ML algorithms, including LR, SVM, KNN, ANN, DT, and RF.

XGBoost was also used for feature selection in[24] to provide a highly efficient and accurate data predictive model in the UNSW-NB15 dataset. Here, the XGBoost produced a subset of 23 out of 39 usable features to distinguish network attack types. The authors could compute the percentage of records in a particular value range corresponding to an attack-type through bi-variate analysis. They also used several ML algorithms to evaluate how XGBoost works on each algorithm. These ML algorithms were Naïve Bayes, Multilogistic Regression, Neural Network, non-linear SVM, and RF. The (Training Accuracy, Testing Accuracy) for those algorithms were achieved as follows: (63%, 64%) with ANN, (56%, 57%) with Multi logistic, (83%, 83%) with Non-linear SVM, (92%, 88%) with XGBoost, (20%, 20%) with Naïve Bayes and (93%, 88%) with RF. Unlike these two studies, XGBoost is not used for data preprocessing in our proposed NDPF. However, as shown in Section 6, our proposed NDPF results in better accuracy.

Using the same dataset, Zong et al.[25] proposed a two-stage classifier approach to investigate network intrusion detection systems based on imbalanced intrusion detection datasets. The proposed approach separates the training and detection of minority and majority intrusion classes, aiming to improve the overall detection rate of minority classes and reduce the error rate. Furthermore, their approach is flexible in the sense that different ML algorithms can be used for each stage of the network intrusion detection systems. However, the authors only evaluated the performance of the RF algorithm, and the accuracy of their approach resulted in 85.78%.

In[26], the authors present an analysis of the relevance of the features in both KDD99 and UNSW-NB15 datasets. The authors use a rough-set theory (RST), a back-propagation neural network (BPNN), and a discrete variant of the cuttlefish algorithm (D-CFA) for their evaluation. First, the RST calculated the dependency ratio between the features and the classes. Then, each feature in the datasets became an input for the BPNN. Finally, they carried out a feature-selection process over multiple runs. Their results showed that a few features in the KDD99 dataset could be used to achieve a classification accuracy above 84%. In addition, some features in both datasets were combined to give higher accuracy.

Tufan et al.[23] proposed anomaly-based ML models to provide a better projection for institutional network. In particular, the authors built and implemented two models, called an ensemble learning model and a convolutional neural network model. The models

were then applied to the UNSW-NB15 benchmarking dataset. Their findings revealed high accuracy rates, the CNN model being slightly more accurate.

In[27], Kumar et al. proposed a novel misuse-based intrusion detection system to detect five categories of threats, such as Exploit, DOS, Probe, Generic and Normal in a network. In their study, the UNSW-NB15 dataset is considered the offline dataset to design its own integrated classification-based model based on which malicious activities in the network are detected. Their data preprocessing consisted of several sub-phases, such as data reduction and feature reduction. The proposed models were tested in UNSW-NB15 and a real-time dataset (RTNITP18) with the different types of threats. Their result showed that the proposed integrated classification-based model is considerably higher than other existing decision tree-based models regarding the accuracy, attack detection rate, mean F-measure, average accuracy, attack accuracy, and false alarm rate.

T. Janarthanan and S. Zargari[28] analyzed features in the UNSW-NB15 dataset to identify the significant features and reduce the number of features in the dataset by using ML techniques, such as RF and Weka, an open-source ML tool. As a result, the irrelevant features are omitted from the dataset, resulting in a faster training and testing process and less resource consumption while maintaining high detection rates. Their proposed subset of features and their findings are comparable with existing works concerning features in the KDD-CUP 1999 dataset.

M. Suleiman et al.[22] evaluated the performance of six ML algorithms including DT (J48), RF, KNN, Nave Bayes, SVM, and ANN on three different types of datasets such as NSL-KDD, UNSW-NB15, and Phishing with aiming to reduce false positive and false negative rate. Their finding was that KNN and DT (J48) were the best performing classifiers regarding detection accuracy, testing time, and false-positive rate. In particular, their archived accuracy were 89.39%, 90.14%, 86.31%, 50.1%, 86.6% and 81.91% for DT (J48), RF, KNN, NB, SVM, and ANN, respectively. Compared to this study, in this thesis, we did not evaluate the performance of the Naive Bayes in the chosen dataset. Nonetheless, the performance of LR is investigated. The accuracy obtained by our proposed NDPF is better and comparable with those of the authors.

Another study that analyzes the performance of several ML algorithms with feature selection on the same chosen dataset was presented in [29]. Chi-Square, a filter-based feature selection technique, is used for data preprocessing to reduce the irrelevant and redundant features. Then, many ML algorithms were evaluated, including KNN, Stochastic Gradient Descent (SGD), DT, RF, LR, and Naïve Bayes in terms of Accuracy, Mean Squared Error (MSE), Precision, Recall, F1-Score, True Positive Rate (TPR) and False Positive Rate (FPR). Their results show that the RF classifier outperforms the other methods in terms of accuracy 99.57%, TPR 0.997, and MSE 0.004. In contrast, the NB shows the highest MSE of 0.234 and lowest accuracy of 76.59% in the selected group of classifiers. In this thesis, our proposed NDPF framework achieves better performance with KNN, DT, RF, and LR.

Intrusion detection in the internet of things (IoT) network using supervised ML algorithms on application and transport layer features using the UNSW-NB15 dataset was presented in [30]. In this study, the authors dealt with the problem of detecting

and removing malicious packets from entering the network. To this end, they proposed feature clusters in terms of Flow, Message Queuing Telemetry Transport (MQTT), and Transmission Control Protocol (TCP) by using features in the UNSW-NB15 dataset. Several supervised ML algorithms, including RF, SVM, and ANN, were implemented on the clusters. Using RF, they achieve 98.67%, and 97.37% of accuracy in Binary and Multi-class Classification. In addition, the authors achieved 96.96%, 91.4%, and 97.54% of classification accuracy in clusters-based techniques by using RF on Flow and MQTT features, TCP features, and top features from both clusters.

N. Moustafa et al.[31] demonstrated the complexity of the UNSW-NB15 dataset in three aspects. First, the authors explained the statistical analysis of the observations and the attributes. Then, they examined feature correlations in the dataset. After that, five ML Naive Bayes, DT, ANN, LR, and Expectation–maximization (EM), were used to evaluate the complexity in terms of accuracy and false alarm rates (FARs). The paper showed that UNSW-NB15 is more complex than KDD-Cup 1999 and is considered a new benchmark data set for evaluating network intrusion detection systems.

In another study[32], the authors investigated the performance of various ML algorithms, including KNN, DT (ID3), LR, SVM, RF, and Adaboost. In their approach, the dataset is first preprocessed using one-hot encoding and normalization. The experimental results showed that ML technology had presented the desired performance in the complex and changeable network traffic data. Their study is somewhat similar to ours, but our proposed NDPF results in a better performance in most ML algorithms.

Md. Islam1 et al.[33] have used a gradient boosting classifier (LightGBM) to perform Binary Classification on UNSW-NB15 dataset. The authors performed preprocessing, feature engineering, and feature selection. Their model achieved 97.21%, 98.33%, and 96.21% with F1 Scores, respectively. Additionally, the model fitted only on train data and achieved a 92.96% F1 Score on the separate test data.

Another evaluation of ML Algorithms for Anomaly Detection is documented in [34]. The authors evaluated twelve ML algorithms in their ability to detect anomalous behaviors. The evaluation was carried out on three publicly available datasets: CICIDS-2017, UNSW-NB15, and the Industrial Control System (ICS) intrusion detection datasets. The results showed that the RF algorithm achieved the best performance in terms of Accuracy, Precision, Recall, F1 Score, and Receiver Operating Characteristic (ROC) curves on all the datasets.

The performance of SVM is shown in [34]. The authors proposed SVM with a new scaling method for Binary Classification and Multi Classification experiments. The performance of the proposed method is evaluated in terms of accuracy, detection rate, and false-positive rate. The accuracy of the proposed method reaches 85.99% for Binary Classification compared to 78.47% by Expectation-Maximization (EM) clustering. For Multi Classification, their proposed SVM method can achieve the testing accuracy of 75.77%, which is 6.17% higher than that of Naive Bayes (NB). In this thesis, we also evaluate the performance of SVM. As can be seen in Chapter 6, the performance of SVM in our proposed framework is superior.

Another study on intrusion detection systems using feature selection for data preprocessing with clustering and four ML algorithms on the UNSW-NB15 dataset is presented in [35]. Here, Naive Bayes, RF, J48, and ZeroR were evaluated. In addition, KNN and EM are used to cluster the UNSW-NB15 dataset into two clusters depending on the target attribute attack or regular network traffic. To develop an optimal subset of features, Correlation-based Feature Selection (CFS) was used, then 4 ML algorithms mentioned above were applied. The result presented that RF and J48 algorithms performed best with 97.59%, and 93.78%, respectively.

In [36], Y. Pacheco et al. evaluated the vulnerability of contemporary datasets, i.e., UNSW-NB15 and Bot-IoT datasets. These datasets represent the modern network environment against popular adversarial deep learning attack methods. Their study showed the feasibility of the attacks for both datasets, where adversarial samples successfully decreased the overall detection performance.

2.3 Anomaly Detection for UNSW-NB15 using Deep Learning

In [37], the authors combined the RF algorithm with the DT algorithm using Anaconda3 and the package management system Conda. In their study, 45 features have been decreased to the most robust four features. The proposed system detects normal data and attacks data with a better accuracy using the Deep Learning technique.

G. Amaizu et al.[38] proposed a network intrusion detection model based on deep neural networks. The model was evaluated in 3 benchmarking datasets: NSL-KDD, UNSW-NB15, and CSECIC-IDS2018. Experiments results in Python Results showed their proposed model performs best for NSL-KDD, followed by UNSW-NB15 and CSECIC-IDS2018, respectively. Model accuracy achieved for these datasets was NSL-KDD (97.89%), UNSW-NB15 (89.99%), and CSE-CIC-IDS2018 (76.47%) was achieved.

In [39], Al-Zewairi et al. used a Deep Learning Binomial Classifier for network intrusion detection System with the UNSW-NB15 dataset. The authors executed three different experiments to determine the optimal activation function, select the essential features, and test their proposed model on testing data. The evaluation results demonstrate that the proposed classifier outperforms other models in the literature with 98.99% accuracy and 0.56% false alarm rate on testing data.

Deep Learning used for Binary Classification is also presented in [40]. The authors investigated the feasibility of an off-line deep learning model by constructing the detection engine with multiple advanced deep learning models. Several ML algorithms were tested on NSL-KDD and UNSW-NB15 datasets. Additionally, the authors developed a TensorFlow-based deep learning library called NetLearner. They then implemented a handful of cutting-edge deep learning models for NIDS. Finally, the authors conducted a quantitative and comparative performance evaluation of those models using NetLearner.

An evaluation of adversarial training on different types of deep neural networks in Deep Learning-based Intrusion Detection Systems (IDS) of deep neural networks is presented in [41]. The authors focused on investigating the effectiveness of different attacks and how to train a resilience deep learning-based IDS using different Neural networks, including ANN,

Convolutional Neural Networks (CNN), and Recurrent Neural Networks (RNN). Their experiments with different deep learning algorithms and different benchmark datasets showed that defense using adversarial training-based min-max formulation increases the robustness of the network.

Chapter 3

Essential Machine Learning Concepts

This Chapter will introduce several fundamental ML concepts to help comprehend the rest of the thesis. We will discuss and describe concepts like Data, Overfitting, Underfitting, Supervised Learning, and Unsupervised Learning, and finally Hyperparameters and the Grid Search Algorithm. The terms mentioned are often used in the thesis. Thereby getting an overview of these sections might be beneficial.

3.1 Data

One of the critical areas to consider in ML is the type of data provided to a model. Without data, ML today would be impossible. A ML system/algorithm has a better chance of understanding and making correct predictions about unknown (real-life) data with a higher quantity. Data are samples or instances from the area that describe the problem we are trying to address. Data in supervised learning consists of samples, each of which comprises an input variable that feeds into an output/target variable that the model will be expected to predict. For example, in classification modeling, the target variable will be a label or a name, whereas, in regression, the target variable will be an integer or floating-point number. The input variable data can be in many different formats, including text, time series, video, and images. Tabular data, often known as structured data, is the most frequent input data, where the data comes in the form of a spreadsheet, a database, or a CSV file (Comma Separated Variable File)[42].

3.1.1 The Four Levels of Measurements

The level at which a variable is measured impacts how we examine the given data. Because variables may be assessed at several levels in many circumstances, we must decide the level of measurement that will be used before beginning data collection. In addition, the four levels of measurement (Figure 3.1) are cumulative, going from lowest to

highest. This implies that they each inherit attributes from previous levels while adding new ones.

- **The Nominal level** implies that the data can be labeled independently into different divisions to categorize it, whereas the categories are unordered.
- **The Ordinal level** implies that the data can be categorized and ranked (ordered), whereas the interval between the data points is unknown
- **The Interval level** implies that the data can be categorized, ranked, and presume equal intervals. Furthermore, no true zero point[1] can be defined on this level.
- **The Ratio level** implies that the data can be categorized, ranked, and include equal intervals, where there is also a true zero point[43].

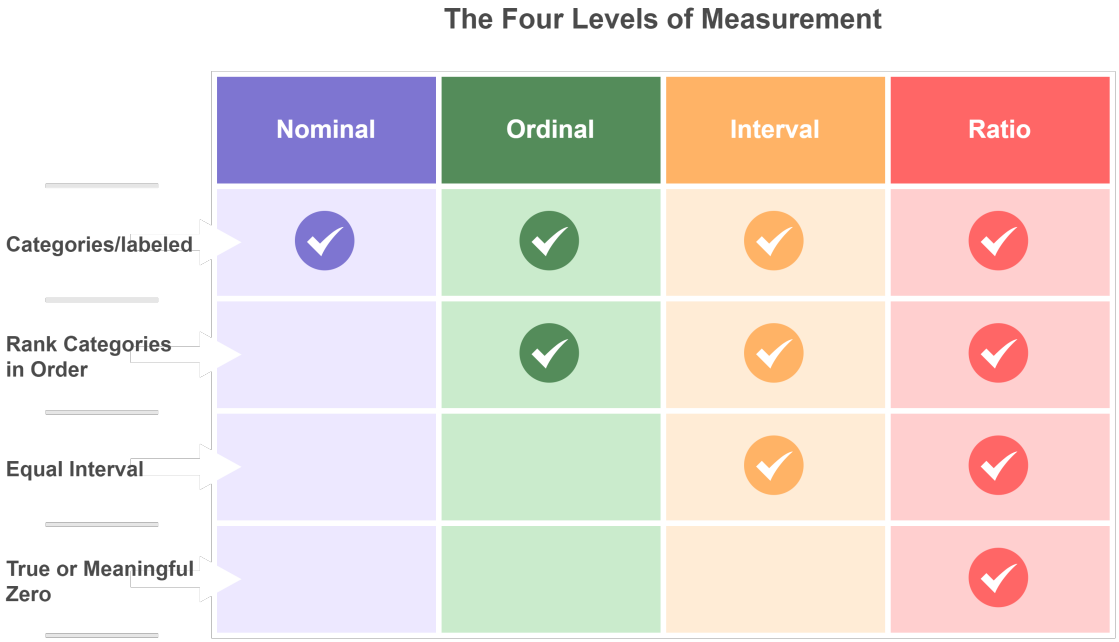


Figure 3.1: The Four Level of Measurements

The Four Primary Data Types

The different data types in ML are essential knowledge to attain for performing Exploratory Data Analysis (EDA)[2] and Feature Engineering. In addition, ML models have different

[1] Real True Zero is a scale on which 0 represents the scarcity of anything. There is no actual zero on an interval scale.

[2] Exploratory Data Analysis (EDA) is a method of analyzing data using visual tools. The main goal is to use graphical representations to uncover trends and patterns and verify hypotheses.

requirements to perform well; therefore, in some cases, variables (features) need to be converted to create acceptable models.

In ML, we can systematize data types into four categories: Numerical-, Categorical-, Time-Series-, and Text Data(Figure 3.2).

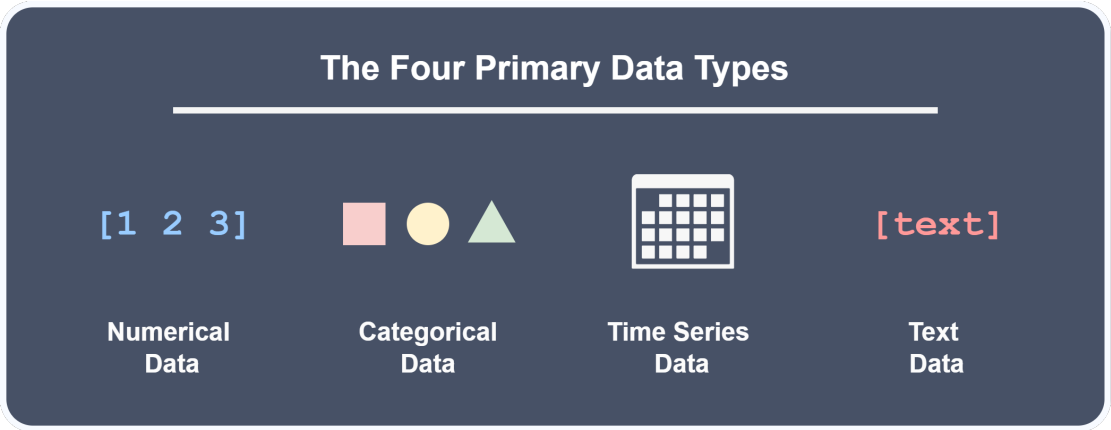


Figure 3.2: The Four Primary Data types

Numerical Data

Numerical data (or Quantitative data) implies that the dataset includes data in the form of numbers (integers and floating-point numbers). Therefore, the data is interpreted as measurements, for example, the number of malware attacks. Numerical data is not sorted or chronologically organized.

Continuous (Interval) and discrete (Ratio) data are two types of numerical data. Discrete data has unique values, but continuous data can take any value within a range. In other words, continuous data has floating-point numbers (numbers with decimal points), whereas discrete data consists of integers (whole numbers)[44].

Categorical Data

Categorical data is a set of information split into different categories. The relationship between categorical data has no mathematical significance. For example, a numerical value of 1 for ”attack”, and 0 for ”non-attack” can neither be added together nor subtracted. Usually, categorical data in datasets are found as a string; therefore, we have to perform one-hot-encoding to transform the categorical data into numerical data for processing. ML algorithms only take numerical values to do mathematical operations and calculations[45].

Categorical Data can be defined into nominal data and Ordinal Data. Nominal data does not include any numerical value, therefore is used to name variables. Nominal data is also referred to as labeled data. Gender, color, name, and hair color are examples of nominal data. The data in Nominal are often acquired through surveys and questionnaires,

as the answers from respondents allow to be written freely. This method does allow for a better conclusion in a study but can cause troublesome and irrelevant material for researchers when they face significant amounts of data[46].

The Ordinal Data type has a predetermined order or rank. The data type is a collection of discrete integers, where the distance between two integers is unknown. For example, the distance between the first and second data points may not be the same as the second and third data points[47].

Figure 3.3 is a comparison of Numerical and Categorical Data.

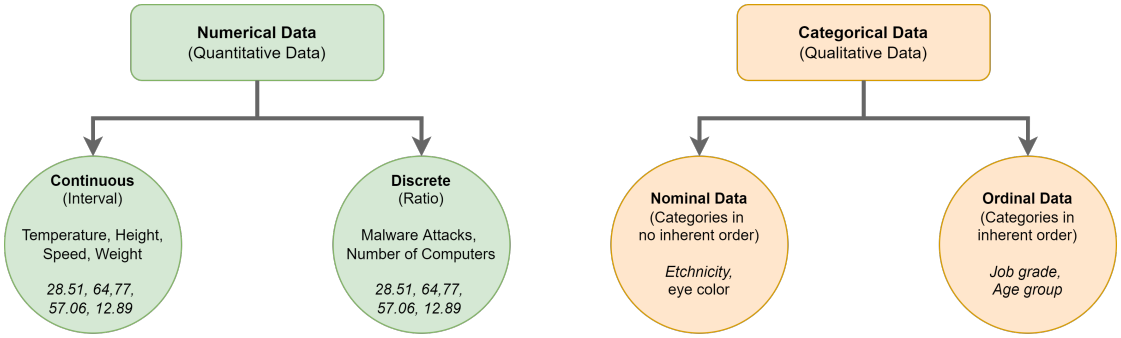


Figure 3.3: Numerical- and Categorical Cata

Time Series Data

Data points referenced at certain times makeup time-series data. Nonetheless, this information is usually obtained at consistent time frames. Using Time Series data allows us to compare minutes, hours, days, weeks, months, and even years. The starting and ending points are fixed, whereas numerical data is an array of numbers that do not relate to any specific time points.

Text Data

We have access to a large amount of text data in posts, articles, and blogs. Various ML algorithms can be employed on such data. Furthermore, various techniques can convert text data into mathematical vectors (vectorization)[45].

3.2 Overfitting and Underfitting

One of the goals of ML is to create a model that performs well on two types of data: training and testing (unseen) data. We want the model to learn from previous examples and apply what it has learned to new situations in the future. Only to measure the model's capacity to generalize to new data do we utilize approaches like a train/test split or k-fold Cross-Validation (refer to Section: 3.4.4 Cross-validation for more information). The two most familiar causes for unsatisfactory ML algorithm performance are overfitting and underfitting [48].

3.2.1 Overfitting

Overfitting data occurs when a model is trained to an extent where it learns the data noise and inaccuracies which impair the ML model's performance on new data (testing/validation data). This implies that the model detects noise or random oscillations in the training data and learns them as new concepts[49]. As a result, the model will have a good performance on training data but poor generalization on new unseen data.

Nonparametric and nonlinear models, which have more adaptability when learning a target function, are more prone to overfitting. As a result, many nonparametric ML techniques incorporate parameters or approaches that restrict and regulate the amount of detail learned by the model. DT (Section: 4.4), for example, are a versatile nonparametric ML technique that is prone to overfitting training data. This issue can be solved by pruning a tree after it has learned to eliminate part of the information it has gathered.

Before we evaluate and run the data, detecting overfitting is nearly impossible. However, when analyzing ML algorithms, there are two key strategies to reduce overfitting.

- Estimate model accuracy using a resampling strategy.
- Save a validation dataset for later.

K-fold Cross-Validation is the most often used resampling approach. It enables us to train and test our models' k-times on different sub-groups of training data, giving us a performance estimate on validation data. A validation dataset is a portion of our complete dataset, which we keep hidden from our model. Firstly, we build and hyperparameter-tune our model relative to the training dataset to assess an overall performance analysis on how well our tuned model performs on unseen data[50].

It is possible to calculate whether the model is overfitting by utilizing bias and variance. Bias is a model's assertions that make a function simpler to learn. The anticipated value will be distant from the actual value if the model is oversimplified, resulting in additional bias. A variance occurs when we train the data on training data and acquire a low error but then change the data and train the same prior model again, resulting in a large error [51]. Whenever a model is overfitting, the values of variance will be high and bias will be relatively low. Let us presume that θ^2 is the point estimator for a parameter or function θ (Equation: 3.1). Therefore, bias is the distinction between the estimator's expected value and the parameter we want to calculate. If the bias is more significant than zero,

the estimator is referred to as positively biased; if the bias is less than zero, the estimator is referred to as negatively biased; and if the bias is precisely zero, the estimate is said to be unbiased. The variance (Equation: 3.2) is defined as the distinction between the anticipated value of the squared estimator and the estimators' squared expectation[52], [53].

$$\text{Bias} = E[\hat{\theta}] - \theta \tag{3.1}$$

$$\text{Var}(\hat{\theta}) = E\left[\hat{\theta}^2\right] - \left(E\left[\hat{\theta}\right]\right)^2 \tag{3.2}$$

3.2.2 Underfitting

Another typical issue in ML is underfitting, which occurs when the model is unable to build a connection between the independent and dependent variables (input and output). Underfitting can be identified by observing the training error the model gives us. The most prevalent indications of underfitting are high bias and low variance. The model underfits, when:

- The model may be unable to draw connections from the dataset due to unclean training data comprising noise or outliers.
- Due to its failure to capture the connection between the input examples and the target values, the model exhibits a large bias. This is most common when dealing with many datasets.
- The model is thought to be overly simple—for example, in complicated settings, we train a linear model.
- Hyperparameters are incorrect. Due to the under-observation of the characteristics, tuning frequently results in underfitting.

We can avoid underfitting by increasing the duration of the training. Oppositely, terminating training too early may cause underfitting. There must be a happy medium where the model maintains a balance of overfitting and underfitting. Adding more features to the model makes it more accurate. This approach will add complexity to the model, improving training outcomes. Another way to avoid underfitting is to remove noise from the training data. Underfitting is frequently caused by the existence of inadequate values and outliers, which may be eliminated by using data cleaning and preprocessing (Section: 5.3) procedures on the data samples[53].

Figure 3.4 visually represents overfitting and underfitting within three ML applications: Classification, Regression and Deep Learning.

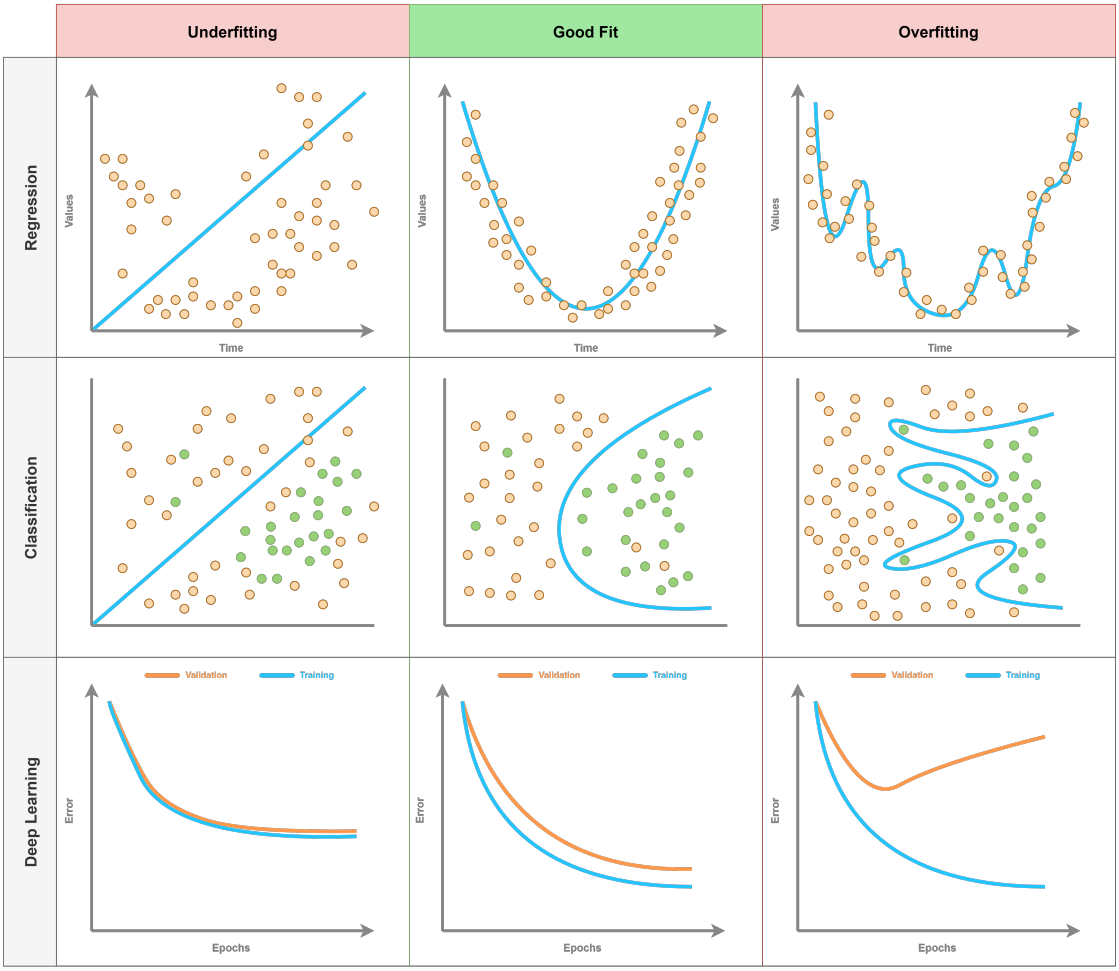


Figure 3.4: Overfitting and Underfitting

3.3 Supervised Learning and Unsupervised Learning

There are two major approaches for implementing ML models: Supervised Learning and Unsupervised Learning. They differ in terms of how the models have developed and the requirements for the needed training data. Since each strategy has its own set of advantages, the project or problem will require either method for solving the issue. Therefore, it is crucial to grasp the fundamental differences between supervised and unsupervised learning as ML becomes increasingly popular. If a company wants to use ML to address an issue, the first step is to comprehend the available data and the situation[54]

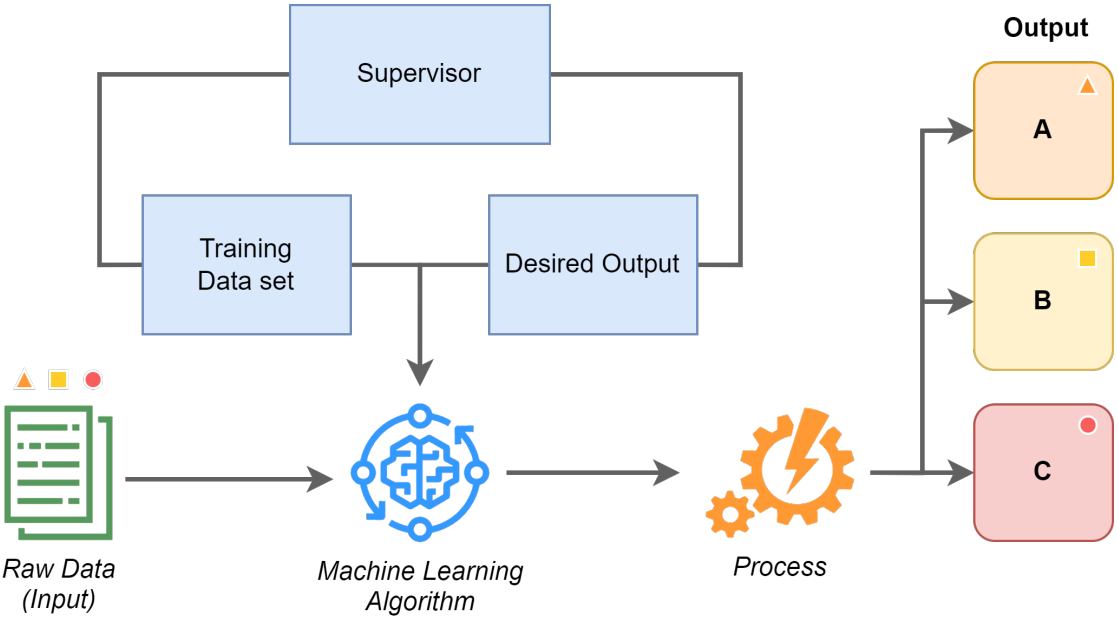


Figure 3.5: Supervised Learning

3.3.1 Supervised Learning

The main objective of supervised learning is to calculate an estimate for a given function (f) based on the input (X) and output (y) features. We refer to this in mathematics as the *Problem of Function Approximation*[52]. The training phase of a given supervised ML model[55] requires labeled input and output variables. During the preparation step, a data scientist may label this training data before it is utilized to train and test the model. In addition, classification can be done on unknown datasets and predict the output once the model has established connections between the input and output data. Because at least some of this methodology involves human control, it is termed supervised ML. The most considerable portion of data is unlabeled and unprocessed. In most cases, human engagement is essential to classify data ready for supervised learning appropriately. Naturally, because vast amounts of precisely labeled training data are required, this may be a resource-intensive procedure.

As a predictive model, supervised ML (Figure: 3.5) categorizes unknown data into recognized categories and forecast patterns, whereas it will learn to identify objects and the properties that categorize them. Supervised ML techniques are frequently used to train predictive models[54].

The supervised learning category includes both regression and classification tasks. Classification and regression issues are the two fundamental domains where supervised ML is beneficial [56]. Each task entails creating a model that learns from training data and can make predictions on new situations for which we have no information. Every

supervised learning method creates relationships between input and output based on training data.

Classification

The process of converting an input value to a discrete value is known as classification (shown in Figure 3.7). Our output in classification tasks is usually in the form of classes or categories. This may include predicting what items or objects there are in an image (Multi-class Classification) or whether or not it will rain today (Binary-classification). A model learns classification from labeled training data, which they analyze and train with. Then, only the features (columns) from the test dataset are utilized to evaluate the performance. Finally, the mapping learned on the training dataset is used to predict the output variable. The training dataset should represent the problem and contain several instances of each class label. Class labels are frequently textual values, such as "attack" and "no attack," and must be converted to numeric values before being fed into a modeling process. Label encoding is assigning a unique integer to each class label, such as "attack" = 1, or "no attack" = 0.

There is no guarantee that one type of classification problem has one algorithm that solves the issue best. This is because the procedure, in many cases, is iterative. Therefore, practitioners are advised to conduct several controlled experiments to determine which algorithm and configuration best perform for a specific classification problem. Classification accuracy is a common statistic for evaluating a model's performance based on predicted class labels. Although classification accuracy is not ideal, it is an excellent place to start for many classification problems. The classification accuracy is the percentage of successfully categorized instances out of all predictions (Equation: 3.3 and 6.1)[57].

$$\text{accuracy} = 100 * \frac{\text{Correct Predictions}}{\text{Total Predictions}} \tag{3.3}$$

There are two primary classification problems we may face:

1. Binary Classification
2. Multi-Class Classification

Binary Classification: When a model can only apply two class labels, it is called Binary Classification. In most Binary Classification problems, one class represents the normal condition, and the other represents the aberrant (abnormal) state, where the normal condition is labeled 0 and the aberrant condition is labeled 1. One of the most popular use cases for this particular classification method is detecting spam emails. For example, based on learning trends of what classifies as spam email, a model may be trained to identify incoming emails as garbage or safe. The most popular algorithm used for Binary Classification consists of LR (Section: 4.2), KNN (Section: 4.6), SVM (Section: 4.3), and DT (Section: 4.4). SVM and LR is developed primarily for Binary Classification since they do not allow more than two classes by default.

Multi-Class Classification refers to classification problems with more than two class labels. Multi-class classification, unlike Binary Classification, does not distinguish between normal and abnormal results. Instead, samples are assigned to one of several predefined classes. Text translation models, for example, are a sort of Multi-Class Classification problem that involves predicting a series of words. In face recognition software, for example, a model may analyze a picture against a large number of possible class labels to identify the person. Multi-Class Classification is often used with RF, DT, Gradient Boosting, KNN, and Naive Bayes. In addition, Multi-Class problems can be solved using Binary Classification algorithms that have been modified, which entails training numerous Binary Classification models for one or many classes. For example, the first method, referred to as "One-vs-Rest": for each class creates a Binary Classification model that compares it to all other classes. The second method is called "One-vs-One": for each pair of classes, create a Binary Classification model. We can modify LR and SVM to apprehend Multi-Class Classification problems. In Figure 3.6 the classification types are illustrated [57].

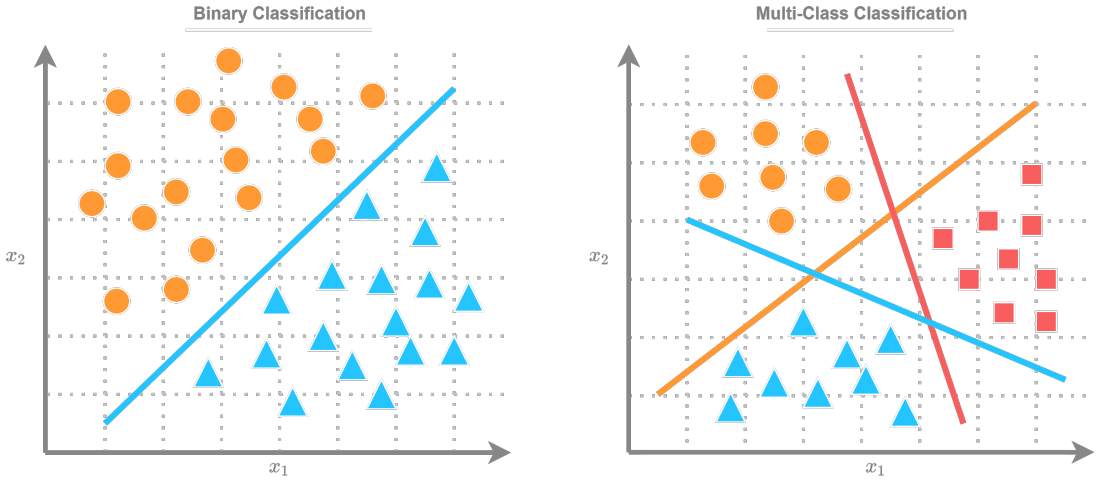


Figure 3.6: Binary Classification and Multi-class Classification

Regression

Another supervised learning approach is regression (shown in Figure 3.7), which employs an algorithm to determine the connection between input (X) and continuous output values (y).

1. A quantity must be predicted in a regression problem.
2. Real-values or discrete input variables can be used in the regression.
3. A multivariate regression issue is a problem with several input variables.
4. If the input variables are sorted by time, the regression problem is called Time Series Prediction.

Since Regression models output a quantity, the performance of the model must be expressed as errors, where the most common way to evaluate is by utilizing Root Mean Squared Error (RMSE), represented in Equation 3.4[54], [58], [59]. There are many real-world applications for regression. One of the fundamental areas for regression is finance, which is utilized to predict stock and training outcomes and movements in the market. Predict the performance of marketing initiatives so that resources may be allocated and refined. Third, predict changes in market value in industries such as retail and real estate. Finally, Predict changes in demographic or geographic health patterns[54]. SVM, LR, Naive Bayes, ANN, KNN, and RF are some of the most used techniques in Supervised Learning[56].

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \hat{x}_i)^2}{n}} \tag{3.4}$$

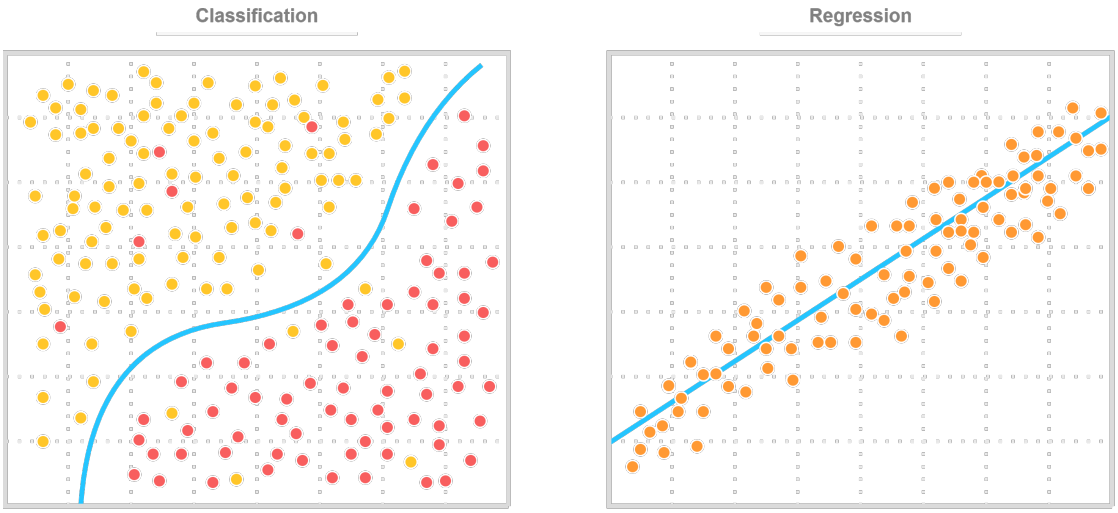


Figure 3.7: Classification and Regression

3.3.2 Unsupervised Learning

ML models utilizing raw and unlabeled training data are known as unsupervised learning[60], [61]. Unsupervised learning is frequently used to detect patterns and trends in raw data or to organize comparable data into categories. Data scientists tend to use this method to understand a dataset better during the early exploration phase.

Unlike supervised ML, unsupervised ML takes a more hands-off strategy. Although a person will configure model hyperparameters like the number of cluster points, the model will analyze large amounts of data efficiently and without human intervention. As a result, unsupervised ML is well suited to answering queries concerning previously unknown patterns and relationships inside data. However, because there is less human

monitoring, the *explainability of unsupervised machine learning*[3] should be given special attention.

The most considerable portion of data is unlabeled and unprocessed. Unsupervised learning is a powerful method for extracting information from data by grouping data by comparable attributes or analyzing datasets for underlying patterns[54].

There are several approaches to this type of learning: *Clustering* (Figure: 3.8) is one of the data mining techniques where it groups unlabeled data relying on similarities. K-means clustering techniques, for example, aggregate related data points into groups, with the K number representing the size and granularity of the grouping. This method is helpful for market segmentation, picture compression, and other applications. Clustering methods include exclusive, overlapping, hierarchical, and probabilistic approaches.

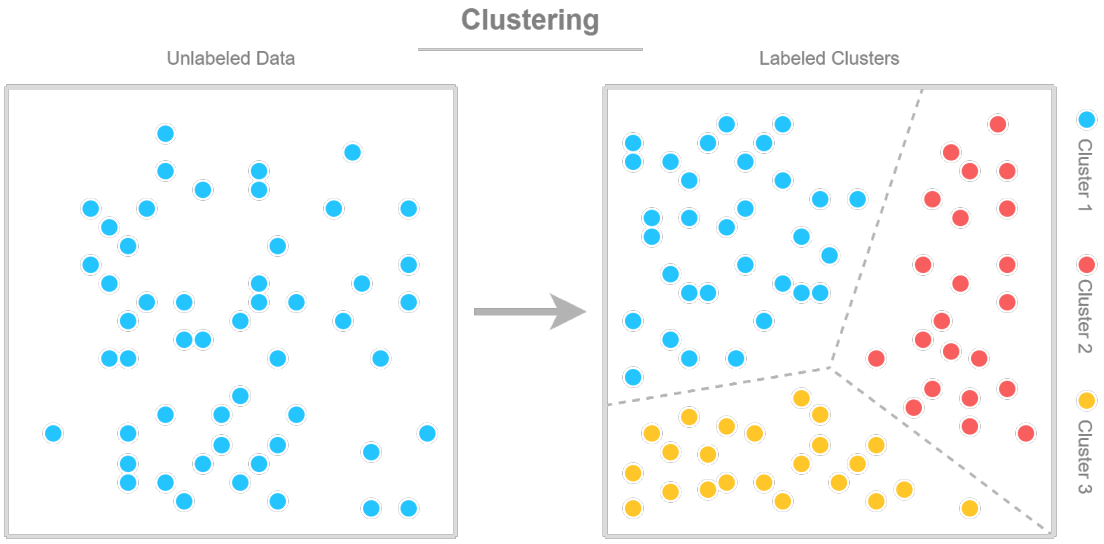


Figure 3.8: Clustering

Another unsupervised learning approach is *association*, which employs several rules to discover associations among variables in a dataset. These strategies are commonly employed in market basket analysis and recommendation engines, such as "Customers Who Bought This Item Also Bought".

When the number of features (or dimensions) in a dataset is too large, *Dimensionality Reduction* (Section: 5.3.3) is a learning approach utilized. It keeps the data integrity while reducing the amount of data inputs to a tolerable size. This approach is frequently employed in the data preparation step (Section: 5.3)[58]. Figure 3.9 represents the most used applications in unsupervised- and supervised learning.

[3]Explainability in ML relates to the procedure of expressing a ML model's conclusion to a human. A person can understand the algorithm and its choice or output if it is model explainable.

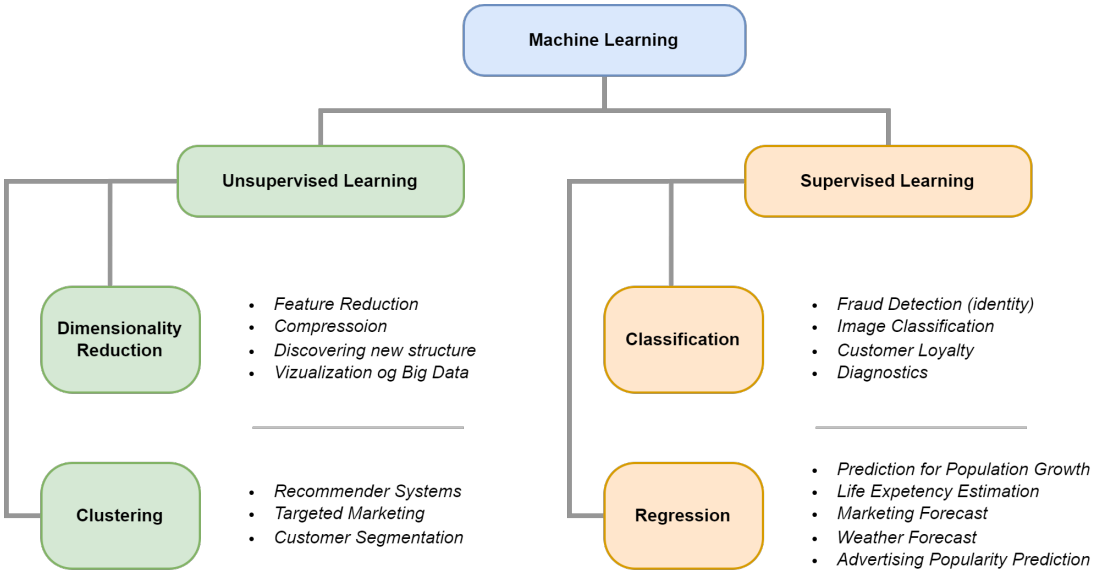


Figure 3.9: Supervised and Unsupervised Learning Use cases

3.4 Grid Search

Grid Search finds the best hyperparameters from a user-defined grid and evaluates each combination using Cross-Validation. To execute Grid Search with Cross-Validation, we can utilize a library from Scikit-learn[62], which imports the Grid Search algorithm. In addition to manual tuning, we use the Grid Search Algorithm, which outputs the most optimal parameters for our dataset. Hence, in this Section, we explain Parameters, Hyperparameters, Cross-Validation, and the Grid Search algorithm.

3.4.1 Parameters

Parameters are the variables used to predict the results of historical data. The algorithm itself sets the parameters with an optimizer which is inbuilt in the algorithm. That means that the parameters do not need to be set by the users of the algorithm. Parameters are part of the training model, and can be the coefficient of variables.

3.4.2 Hyperparameters

Hyperparameters are the variables that the user sets before executing the model. Depending on the model, the variables significantly impact the model's accuracy, both negatively and positively. Hyperparameters are a way to evaluate the parameters of the model. Examples of hyperparameters can be C and gamma in SVM, K in KNN, and max depth and criterion in RF and DT.

3.4.3 Hyperparameter Tuning

Most ML algorithms need to do hyperparameter tuning to get satisfactory results from the model. There are several ways to tune the hyperparameters for every algorithm, where one of the methods is a manual search. When doing a manual search, it goes by the trial and error method, which can be time-consuming. In addition, it can be challenging to know if there is a better combination of hyperparameters. Another method used by researchers is Random Search (RS). The algorithm requires a set of hyperparameters defined by the user, and then it executes the parameters in a random combination of a amount x, where the user also determines x.

Grid Search takes all the combinations in the grid, with the user specified grid, and calculates the performance for each iteration. But since the Grid Search algorithm iterates through every combination, the whole process is very computational heavy and time-consuming. So this will limit the Grid Search since it will only look through the variables the user specified, where involving more values into the grid will make the algorithm more time consuming.

3.4.4 Cross-Validation

The dataset is split into training and testing sets when working with a ML algorithm. Then the Cross-Validation will further divide the training set into train data and validation data. The reason the algorithm does this is to validate itself. A popular Cross-Validation method is the K-fold Cross-Validation [63]. The K-fold Cross-Validation splits the process up into K partitions. Every iteration will use $K - 1$ partitions for training and one for testing. For each iteration, the partition used for testing will be pushed one step further, as shown in Figure 3.10, and it will use the rest of the partitions for training. Every test partition will record its performance, and an average score will return to the user. As a result, executing the K-fold Cross-Validation for every hyperparameter combination will be time-consuming

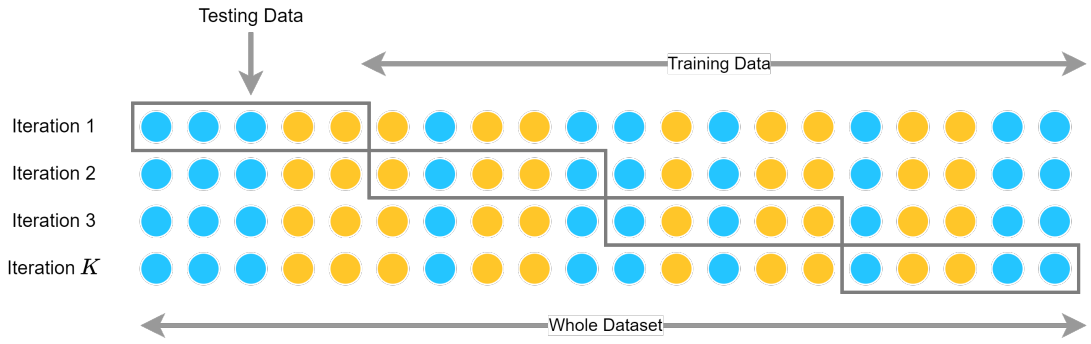


Figure 3.10: K-fold Cross-validation

3.4.5 How To Use Grid Search

The gridsearchCV method[4], is the Grid Search in combination with Cross-Validation, which is available in the Scikit-learn library. The method is initialized by calling `GridSearchCV()`. Grid Search primarily has four arguments to take into consideration and is explained in Table 3.1.

Table 3.1: Hyperparameters for the Grid Search

Hyperparameter	**Description**
Estimator	The estimator is the model it will find the hyperparameters for, such as SVR for SVM and DT for the decision tree.
Param_grid	A Python dictionary with the names of the hyperparameters and a list of the parameters as a value.
Scoring	Scoring is how it evaluates the performance of the test set.
CV	CV determines the strategy of the Cross-Validation. It can be an integer to determine the amount of K for the K-fold Cross-Validation, CV splitter, or some custom function.

Grid Search does not necessarily have to be used on one hyperparameter at a time. It can work on several hyperparameters simultaneously to get the best combination for accuracy. When the Grid Search process is over, the best accuracy and hyperparameters can be fetched with inbuilt methods from the library.

[4] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html`

Chapter 4

Machine Learning Algorithms

4.1 Artificial Neural Network (ANN)

Artificial Neural Networks (ANN), often known as neural networks, are a subset of ML algorithms that are at the heart of deep learning algorithms. A neural network's core concept is to imitate a large number of tightly interconnected brain cells inside a computer so that it can learn new things, identify patterns, and make decisions in a human-like approach. A neural network does not need to be explicitly designed; it learns on its own, just like a brain.

ANNs include an input layer, one or more hidden layers, and an output layer, each of which links to the layers on either side. They range from a few dozen to hundreds, thousands, or even millions of artificial neurons known as units or nodes. The connections between two units are represented by a weight, which might be positive (if one unit excites another) or negative (if one unit inhibit another). The more significant the impact of one unit on another, the higher the weight. This is similar to how synapses in the brain allow brain cells to communicate. If a node's output exceeds a threshold value, the node is activated and sends data to the next layer of the network. Otherwise, no data is sent to the network's next layer.

Various aspects influence how a neural network operates: Hyperparameter tuning (i.e., Learning rate & activation function), dataset size, and model complexity. Furthermore, Neural networks use training data to learn and increase their accuracy over time. However, once these learning algorithms have been fine-tuned for accuracy, they become effective tools in computer science and artificial intelligence, allowing us to categorize high-density data quickly. For example, human specialists' manual identification, speech recognition, or picture recognition tasks can take minutes rather than hours. Figure 4.1 represents a generic Neural Network model.

4.1.1 Epoch & Batch Size

An epoch is a complete transit of the training data through the algorithm in ML. The algorithm's epoch value is a significant hyperparameter (refer to Section 3.4.2, for more

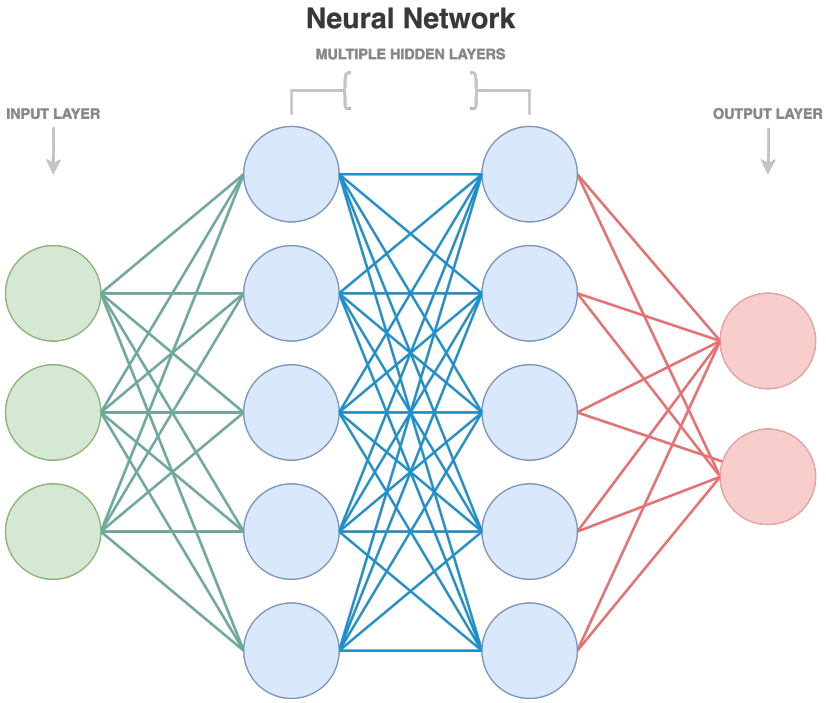


Figure 4.1: Generic Neural Network Model

information). It denotes the number of epochs or complete runs through the algorithm's training or learning phase for the whole training dataset. With each epoch, the dataset's internal model parameters are changed. As an outcome, a single batch epoch is named after the gradient learning algorithm. The batch size of an epoch is usually one and is always an integer.

The number of epochs equals the number of iterations, and the batch size equals the whole training dataset. This is rarely the case for practical reasons. More than one epoch is used in several models. In general, the relationship is $d \times e = i \times b$, in which d is the dataset size, e is the number of epochs, i is the number of iterations, and b is the batch size.

4.1.2 Weights and Bias

Weights are actual values associated with each feature, and they indicate how important the feature is in predicting the final result. The relevance of each feature is represented by the weights associated with it in predicting the output variable. Features with weights close to 0 are deemed less critical in the prediction process than weights with values of greater magnitude

Bias (or a constant vector) shifts the activation function towards positive or negative values. In the line equation, the bias is known as the y-intercept. The neural network merely performs a matrix multiplication on the inputs and the weights if the bias is not

included. Therefore, this can easily lead to the dataset being overfitted. Furthermore, since bias is the inverse of the threshold, bias determines when the activation function is activated.

4.1.3 Layers

The input layer (the initial data for the network, usually noted as vector X) accepts various types of data from the outside world, which the network will attempt to learn about, identify, or process. Then, the data is passed to the rest of the network from this layer. These input variables can be imported from external sources such as web services (API) or text-formatted files such as CSV.

The hidden layer serves as a bridge between the input and output layers and is accountable for neural networks' high performance and complexity. Moreover, they may do various tasks simultaneously, such as data transformation and automated feature generation. The neurons add the bias and determine inputs and weights before performing an activation function.

The output layer is in control of generating the final product. Therefore, there must always be one output layer in a neural network. The output layer receives the preceding values from the above layers (hidden layers), executes the computations using its neurons, and then produces the output.

4.1.4 Neurons

The neurons, shown in Figure 4.2, add the bias and determine the weighted sum of inputs and weights before performing an activation function. A neuron can be addressed as a container with several components:

1. Activation function
2. Input variables
3. Vector of weights
4. Bias

When a neural network is trained on a training set, it is initially given a set of weights. The ideal weights are then created by optimizing these weights during the training phase.

The first step of the neuron is to calculated the weighted sum of the inputs, shown in Equation 4.1.

$$Y = \sum(weight * input) + bias \tag{4.1}$$

We can define the inputs in general as $x_1, x_2, ..., x_n$, and the weights as $w_1, w_2, ..., w_n$. The weighted sum is then calculated as in Equation 4.2.

$$x_1w_1 + x_2w_2 + ... + x_nw_n \tag{4.2}$$

After that, the weighted sum is given a bias which is a constant, shown in Equation 4.3.

$$x_1w_1 + x_2w_2 + ... + x_nw_n + bias \tag{4.3}$$

Finally, the computed value is sent to the activation function (AF), which generates a result, shown in Equation 4.4.

$$AF(x_1w_1 + x_2w_2 + ... + x_nw_n + bias) \tag{4.4}$$

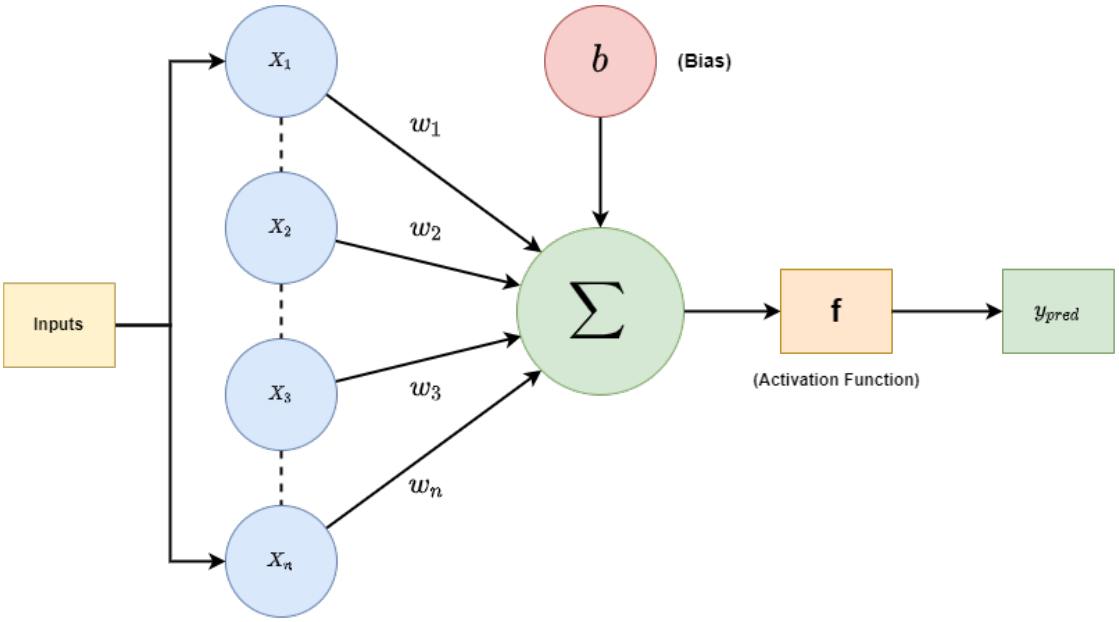


Figure 4.2: Artificial Neuron

4.1.5 Cost/Loss Function

The cost function is a method of assessing the overall performance of our given ML algorithm or model. It compares the model's projected and actual outputs to determine how far off the model was in its predictions. If our predictions differ significantly from the actual numbers, it returns a more significant number. The cost function indicates how well our model has improved as we modify it to enhance predictions.

The term "loss" in ML refers to the difference between the actual value (x-input), and predicted (y-output) value. The Loss function measures the Loss and outputs the actual number during the training phase. This particular function is used in many supervised ML algorithms since they depend on it.

We can differentiate the Loss- and Cost function: We use the term Loss function when referring to the error for a single training sample. However, when referring to the average value of the Loss functions across a complete training dataset, we use the term Cost function.

We want to optimize the value that the Cost function offers us. Where y represents the desired output and $\hat{y}$ represents the actual predicted output from a neural network. We start summing from the first case $i = 1$ and work our way through the squares of the differences between the desired output y and the expected output $\hat{y}$ for each observation for each sample.When, a ratio between the weights (and biases) and the cost function is calculated, the ones with the highest ratio will significantly influence the cost function and provide us with the most worth.

In ML, there are several cost functions, each with its own set of applications depending on whether the task is a regression or classification[64].

1. Regression
2. Binary Classification
3. Multi-class Classification

Regression Cost Functions

Regression is a method for examining the relationship between the input and output variables (independent and dependent variables). In ML, it is utilized as a method for predictive modeling, whereas an algorithm is employed to predict continuous values. The cost function in Regression is referred to as the "Regression Cost function." The error is calculated between the actual input and the predicted output based on distance, with Equation 4.5, Where y is the independent variable (input), and y' is the dependent variable (output). Figure 4.3 represents a regression line with the distance of error.

$$Error = y - y' \tag{4.5}$$

Mean Error (ME)

The equation for Mean Error (Equation 4.6) stands as a fundament for other regression cost functions. First, the total mean error for each training data sample is computed. Then, the final output can cancel each other out when added together, resulting in a model with a mean error of zero. This implementation is not recommended; hence, it lays the groundwork for alternative regression cost functions.

$$ME = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i) \tag{4.6}$$

Mean Squared Error (MSE)

The Mean Squared Error (MSE)is a metric that calculates the distance to a given set of data points relative to a regression line. A large MSE value means that the given data points are distributed widely around the regression line (mean), whereas a smaller MSE value means that the given data points are closer to the regression line. A smaller MSE value is preferred since it represents data values that are not skewed and includes fewer errors. Because we are continually squaring the errors, the MSE will not be negative.

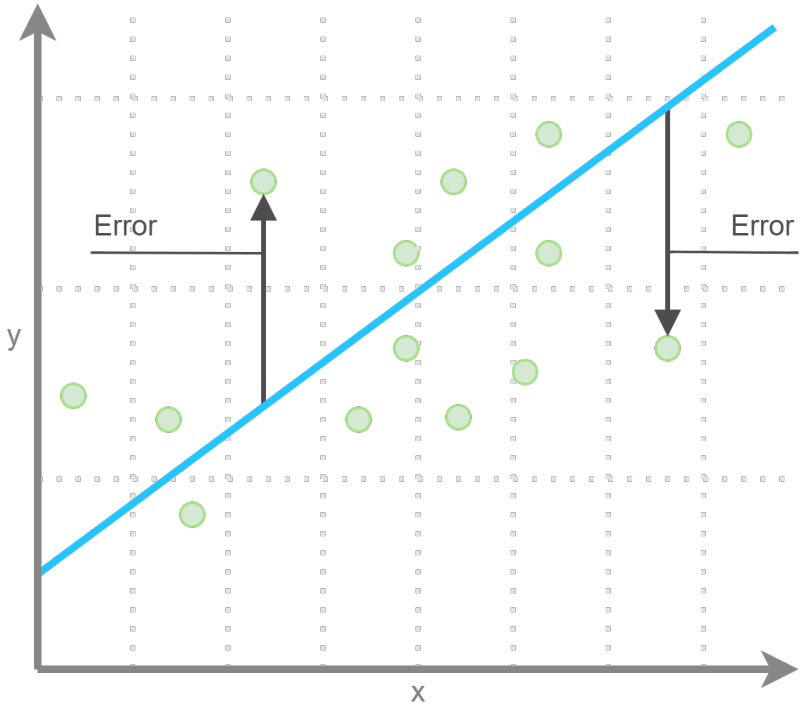


Figure 4.3: Regression Line with Error

The MSE is mathematically defined by Equation 4.7, where the notations are listed in Table 4.1. The code implementation in Python, is presented in Listing 4.1.

The main advantage is that the MSE function helps assure that our trained model does not include any predictions on outliers with significant errors. Furthermore, since MSE squares numbers, the function puts more weight on the errors. One disadvantage is that the squaring section of the equation magnifies the error if our model produces just one error. In most cases, we do not worry about the outliers on a dataset and want a well-performing all-around model[65].

Table 4.1: Mean Error Notations

Notation	**Explanation**
n	The number of total data points
y_i	Observed Values
$\hat{y}_i$	Predicted Values

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{4.7}$$

```python
import numpy as np

# Loss Function - Mean Squared Error
def loss_mse(y_pred, y_true):
    squared_err = (y_pred - y_true) ** 2
    sum_squared_err = np.sum(squared_err)
    loss = sum_squared_err / y_true.size
    return loss
```

Listing 4.1: Mean Squared Error implementation in Python

Mean Absolute Error (MAE)

The Equation 4.8, for Mean Absolute Error (MAE), is only scarcely different from MSE. The MAE will not be negative in any cases like the MSE since we are applying the absolute value of the output. Therefore, the MAE directly has an advantage over the MSE's disadvantage, meaning: that all of the values (errors) will be defined on a linear scale since we are taking the absolute value. As a result, the MAE function measures our models' performance evenly by not putting as much weight on the outliers as the MSE does. Python Code implementation is shown in Listing 4.2

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{4.8}$$

```python
import numpy as np

# MAE loss function
def loss_MAE(y_pred, y_true):
    absolute_error = np.abs(y_pred - y_true)
    sum_absolute_error = np.sum(absolute_error)
    loss = sum_absolute_error / y_true.size
    return loss
```

Listing 4.2: Mean Absolute Error implementation in Python

Huber Loss

We now know that the MSE excels at detecting outliers, whereas the MAE ignores them. Therefore, the MSE function and MAE functions are combined in the Huber loss function to create something applicable in both situations. We define the Equation 4.9 for Huber loss with δ and δ^2, to make the equation differentiable to keep the derivatives continuous since the use case of Huber loss is connected to derivations (i.e., Gradient Descent).

According to Equation 4.9, the MSE should be used for loss values less than δ, while the MAE should be used for loss values more than δ. This successfully combines the finest qualities of both loss functions.

When we use the MAE for more significant loss values, we reduce the weight we give outliers while still getting a well-rounded model. In order to keep a quadratic function in the center, we apply the MSE for the more diminutive loss values. The Python implementation for Huber Loss is presented in Listing 4.3.

$$L(\delta, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & for|y - f(x)| \leq \delta \\ \delta|y - f(x)| - \frac{1}{2}\delta^2 & Otherwise \end{cases} \tag{4.9}$$

```python
import numpy as np
# Huber Loss Function
def huber_loss(y_pred, y, delta=1.0, n=data_points):
    mse_huber = n*(y - y_pred)**2
    mae_huber = delta * (np.abs(y - y_pred) - n * delta)
    return np.where(np.abs(y - y_pred) <= delta, mse_huber, mae_huber)
```

Listing 4.3: Huber Loss Function implementation in Python

Figure 4.4 illustrate the difference in MAE, MSE and Huber Loss correspondingly.

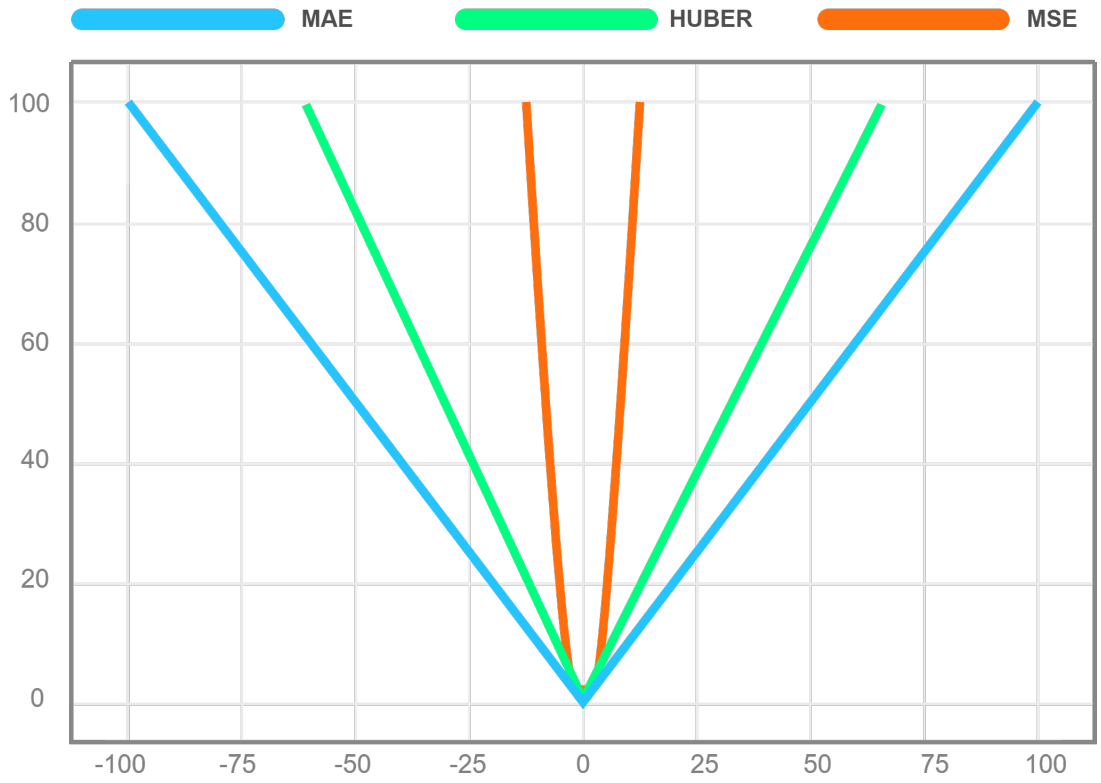


Figure 4.4: Mean Squared Error and Mean Absolute Error

Binary Classification Cost Functions

Classification Cost functions diverge from regression cost functions. For example, Cross-entropy Loss is a well-known Loss function for classification. Cross-entropy, also known as Log Loss, is utilized to measure the classification efficiency, where the output (probability) can range between 0 and 1. As the projected probability differs from the actual label,

Cross-entropy Loss grows. Unlike other Loss functions, Cross-entropy computes the cumulative entropy across the given distributions[66].

Equation 4.10 represents Binary Cross-Entropy, where p_i is the probability of class 1, whereas the probability of class 0 is defined as $1 - p_i$. Whenever a class value is 0, the relevant part will remain, and the other section will disappear. The Equation 4.11 can be used for Multi-Class Classification, where n represents the number of rows, and m represents the number of classes. Figure 4.5 illustrates Cross-entropy where $y = 1$ (True Label)[67], and the Python implementation for Cross-entropy is presented in Listing 4.4.

$$BCE = \frac{1}{n}\sum_{i=1}^{n} - (y_i \times \log(p_i) + (1 - y_i) \times \log(1 - p_i)) \tag{4.10}$$

$$BCE(\text{Multi-class}) = -\frac{1}{n}\sum_{i}^{n}\sum_{j}^{m} y_{ij}\log(p_{ij}) \tag{4.11}$$

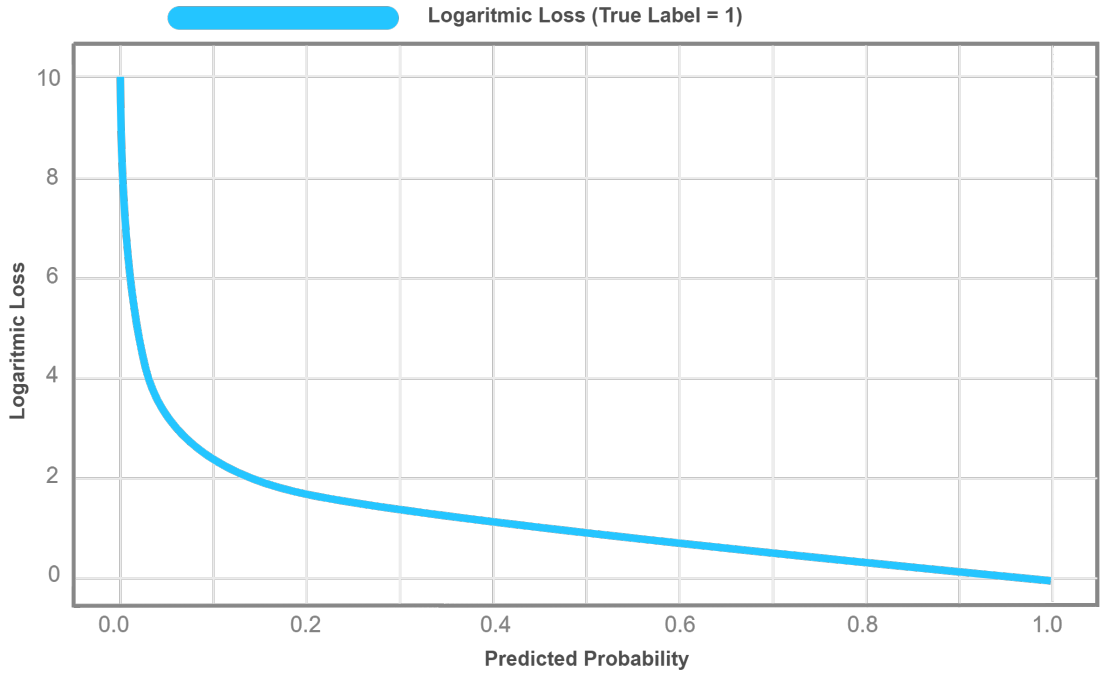


Figure 4.5: Cross-Entropy

```python
# Calculates the Cross Entropy
def Cross_Entropy(y_hat, y):
    if y == 1:
        return -log(y_hat)
    else:
        return -log(1 - y_hat)
```

Listing 4.4: Cross Entropy implementation in Python

Gradient Descent

Gradient descent is a widely used optimization algorithm, and it is the most typical method for optimizing neural networks by locating a function's local[1]- or global minima[2], as shown in Figure 4.6[68].

The Gradient Descent is a first-order optimization technique that allows a model to learn the distinctions between the actual label and the predicted label (y and $\hat{y}$) by minimizing error in the sense of correcting the path of the gradient. As the model iterates, the learning rate (η) determines the level of the step we take each iteration to attain the local- or global minimum. The learning rate shown in Figure 4.8, is a positive constant used to specify the size of the step in the gradient descent algorithm. We use the subtraction sign to shift the weight vector w_i in a direction that reduces E (Cost function)[69]. Additional tweaking of the parameters results in little or no change in the loss. This process is known as convergence, shown in Figure 4.7.

The Gradient Descent algorithm begins by initializing a vector with a random weight, where it gradually updates the stages taken to find a vector (w) that aims to minimize the error (E). The negative of this vector shows the direction with the highest decline, as the gradient defines the direction that generates the steepest increase in Error. The weight vector (w) is changed in the direction that creates the steepest drop and the error at each step.

We need an efficient approach to calculate the gradient at each step to build a fitting algorithm that repeatedly updates its weights. By differentiating the Cost function (E), we can calculate the gradient vector that can be partially derivated. We can express the update rule for Gradient Descent (Delta Rule) by Equation 4.12, where corresponding notations are presented in Table 4.2. We will put the equation into practice in Section 4.1.6[70].

Table 4.2: Notations - Gradient Descent

Notation	**Explanation**
d	Data point
Δw	New Weight vector
w_{old}	Old weight vector
E	Error
η	Learning rate

$$\Delta w = w_{old} + \eta \Delta w \tag{4.12}$$

[1] **Local Minima:** A function's local minimum is a position where the function value is less than at adjacent points but may be higher than at a distant point.

[2] **Global Minima:** A global minimum is a point where the function value is less than all other possible values.

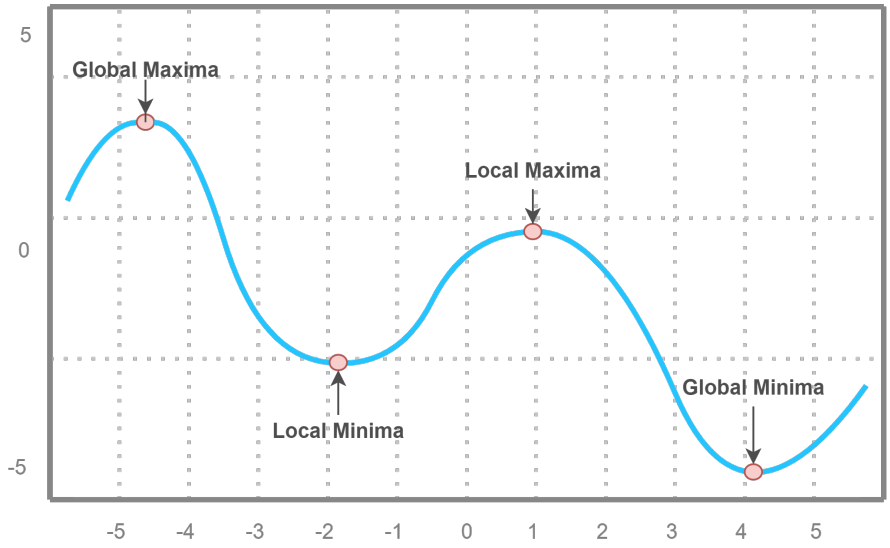


Figure 4.6: Local/Global Minima/Maxima

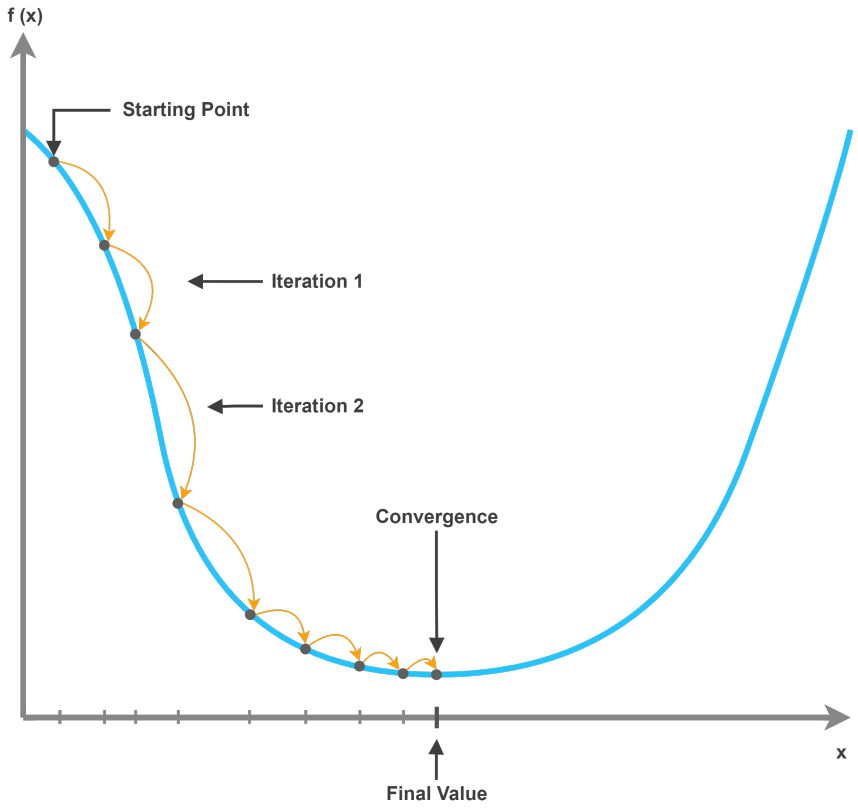


Figure 4.7: Convergence

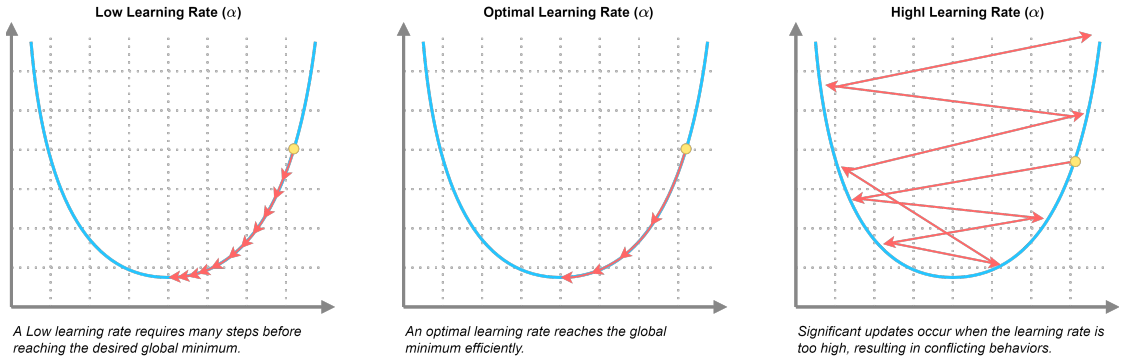


Figure 4.8: Learning Rate η

4.1.6 Feedforward- and Backpropagation

Firstly, let us define a Feedforward Neural Network (FNN). A Feedforward Neural Network is an ANN where the nodes are not connected in a cycle. Instead, the information flows only in a forward direction, where the input from the model makes calculations regarding the hidden layer to generate a final output. The single-layer perceptron (SLP) is the most basic form of an FNN. In an SLP, the sequence of inputs feeds into the layer and gets multiplied by the weights.

The weighted input values are then summed together to form a total value. If the total of the values is more than a predetermined threshold, which is generally set at 0, the output value is usually 1, and if the total is less than the threshold, the output value is typically -1. The SLP is a famous feedforward neural network architecture frequently used for classification. SLPs can also contain ML features. The neural network utilized the delta rule to compare the outputs of the nodes intended values. This results in weight adjustment in the training phase to create a more precise output. This training and learning process results in gradient descent. The technique of updating weights in Multi-Layered Perceptrons is virtually the same. However, the process is referred to as backpropagation. In such circumstances, the output values provided by the final layer are used to alter each hidden layer inside the network[71]–[73].

Feedforward in Neural Networks

Forward propagation is vital in Neural Networks since it helps determine whether provided weights are suitable for training on the problem[74]. There are two main stages in forwarding propagation:

Stage one: Summing the products. It entails multiplying the weight vector (w) by the input vector. This process is recursive until we reach the last layer. It is in the last layer that decisions are made.

Stage two: Feed the summation of the products from stage one into an Activation function. Every layer receives the sum of the weight and input vector to produce the

output layer. The output of one layer is then sent into the input of the subsequent stage, which is multiplied by the weight vector in that layer. This process is recursive until we reach the activation function in the output layer.

Figure 4.9 illustrates the process of Forward propagation.

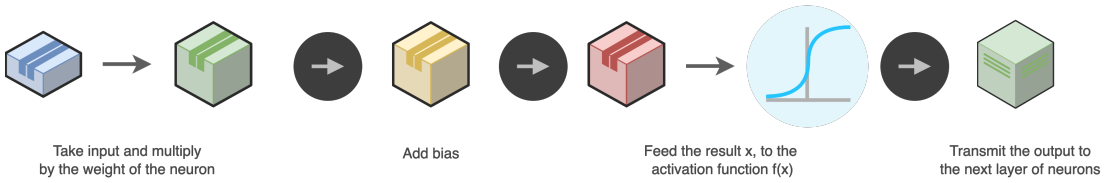


Figure 4.9: Forward Propagation

Backpropogation in Neural Networks

Backpropagation is a shortened form for ”backward propagation of errors” in a neural network. It is a standard way to train ANNs. The basic concept of backpropagation was firstly introduced, in the context of flight performance optimization, by Henry J. Kelley in 1960[75], but the importance was not entirely noticed until a paper published in 1986 by David E. Rumelhart, Richard Durbin, Richard Golden, and Yves Chauvin[76]. As we know by today, backpropagation helps calculate the gradient of a Loss function with respect to the network's weights. In a nutshell, backpropagation computes the gradient in the weight space, allowing us to tune the weight in such a way that we minimize the error in the prior epoch. By fine-tuning the weights, we may lower error rates and improve the model's generalization, making it more dependable[77].

We always start at the output layer and work our way backward, tweaking weights and biases. We can only modify the weights and biases; however, activations are direct calculations of those weights and biases, so we can implicitly adjust every aspect of the neural network to obtain the desired output. The input layer is an exception since this is where the dataset is allocated. The goal is to determine how the Cost function changes in response to a certain weight, bias, or activation. We employ partial derivatives in mathematics because it enables us to compute relationships between neural network components and the Cost function we seek to minimize. We can successfully modify the necessary weights (w) and biases (b) to lower the cost function once we know what influences it. In a mathematical sense, we calculate the partial derivates of the Cost function, with respect to the weights and bias[70], [78]–[81].

Let us for instance take a simple architecture of a Multilayer Perceptron (MLP), as shown in Figure 4.10, which in our case is a fully connected feed-forward Neural network, including only one hidden layer. The notations used for backpropagation is listed in Table 4.3.

Table 4.3: Backpropagation Notations

Notation	Explanation
$i \leftarrow \mathrm{i}_1, ..., i_i$	Input node
$j \leftarrow \mathrm{h}_1, \ldots, h_j$	Hidden node
$k \leftarrow \mathrm{o}_1, \ldots, o_k$	Output node
w_{ji}	The weight for connection between node i and j
w_{kj}	The weight for connection between node j and k
d	Target value for each ouput node
$(x_1, d_1), \ldots, (x_l, d_l)$	Training set with coupled with input and target $\leftarrow (input, target)$

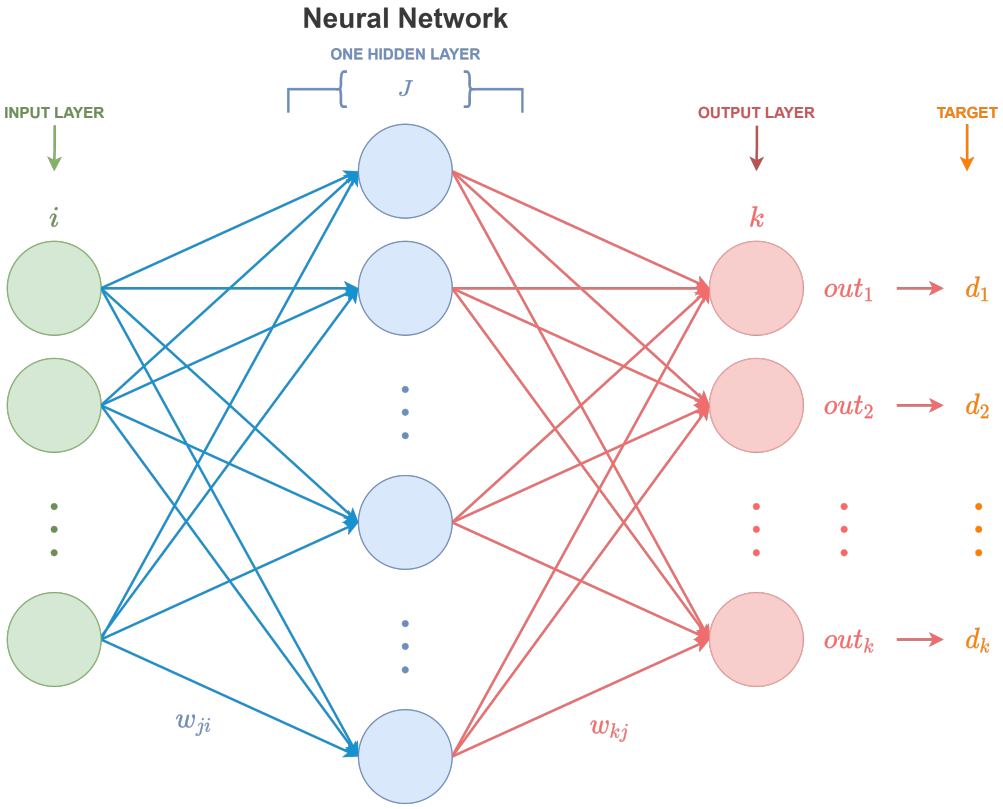


Figure 4.10: The Multilayer Perceptron in our case

We utilize the Mean Squared Error presented in Equation 4.7 in Section 4.1.5 to minimize the cost-function for backpropagation by adjusting the bias (b) and weights (w), where the value returned from the Cost function is often referred as cost or error.

We determine the adjustment given by the gradients of our cost function and get as low error/cost as possible. The properties of MSE make it a good fit for calculating the error in backpropagation since we are relying on derivatives.

However, before we redefine the equation, we have to make two assumptions for our Cost function to implement backpropagation.

1. We can express the Cost function as an average of Cost functions for every individual training sample. We refer to this as "patterns". We will also assume that the pattern p is a constant, where E_p can also be written as E

2. We can write the function of the output given by the neural network.

We can redefine equation for MSE (Equation 4.7) as Equation 4.13, where E_p is the error for every single pattern p

$$E_p = \frac{1}{2}\sum_k (d_k - o_k)^2 \tag{4.13}$$

We calculate the difference between the actual target d and the generated output o for pattern p, with respect to the output units k, we can then define the total error for all patterns in Equation 4.14.

$$E_{total} = \sum_p E_p \tag{4.14}$$

The model can calculate how much to alter the parameter for the function to come near the predicted output d, which is based on the value for the error function. Backpropagation applies three differentiation rules we need to consider: Chain Rule (4.15), Composition of functions (4.16) and the Product Rule (4.17).

$$\frac{\delta f(x)}{\delta x} = \frac{\delta f(x)}{\delta g(x)} \times \frac{\delta g(x)}{\delta x} \tag{4.15}$$

$$\Delta(f(g(x))) = (f \odot g) = f'(g(x)) \times g'(x) \tag{4.16}$$

$$\Delta(f(x) \times g(x)) = f'g(x) \times g + g'(x) \times f(x) \tag{4.17}$$

We can minimize the Cost function using **Gradient Descent** (Figure: 4.11) in backpropagation. The algorithm, as mentioned in Section 4.1.5, iteratively derivates and works through the model to find the global minimum. As a result, our goal is to calculate the error's gradient and then take the inverse to get the direction the algorithm has to descent on E_{total}[70].

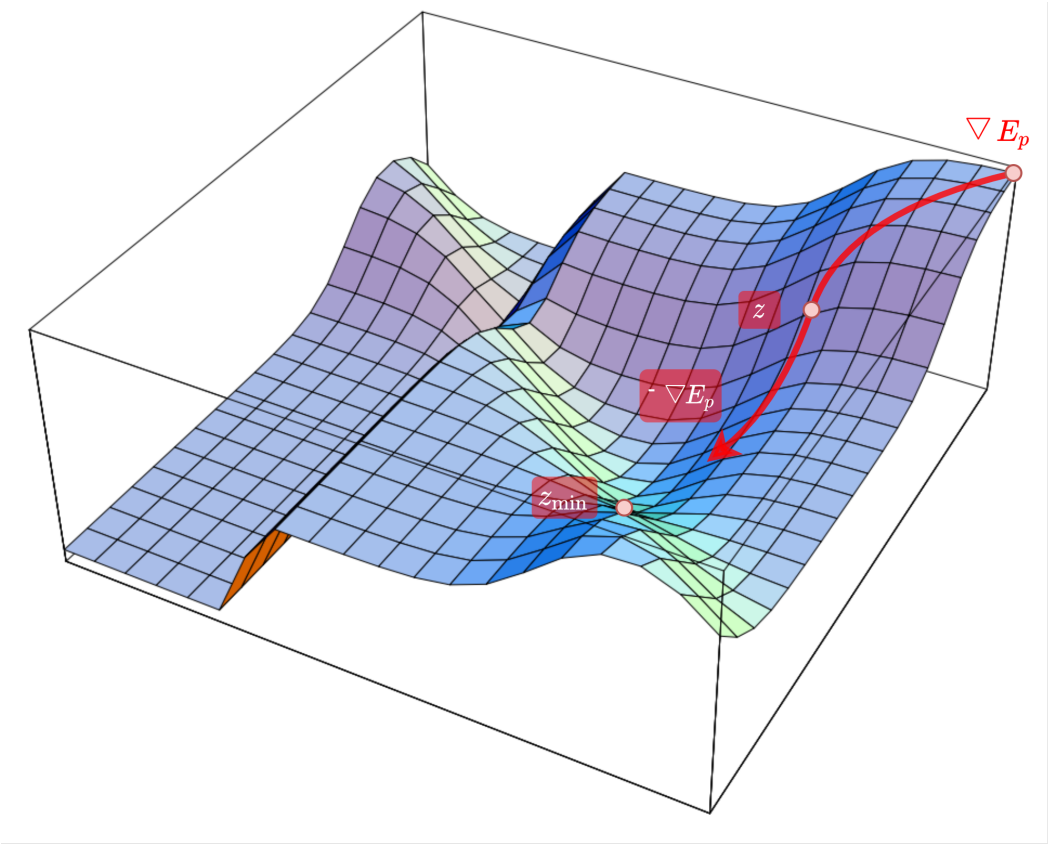


Figure 4.11: Gradient Descent

We had earlier defined Equation 4.12 for gradient descent. Let us now redefine the equation so we can compute for each pattern p given by our model. The computation of Gradient Descent for our example case is given by algorithm 1. The goal is to compute the Cost function E_{total} by calculating the gradient and updating the weights in our network. This particular process must be repeated iteratively until we achieve convergence or meet the criteria[70].

$$\Delta w = -\frac{\delta E_{total}}{\delta w} = -\sum_{p} \frac{\delta E_p}{\delta w} = \sum_{p} \Delta_p w \tag{4.18}$$

We can now make the calculations for a generic unit t, which can either be an output or a hidden unit (node). Firstly we can compute the derivative of error for a given pattern p, with our defined generic unit t.

$$\frac{\delta E_p}{\delta w_{ti}} \tag{4.19}$$

Algorithm 1: Gradient Descent

Data: Weights w, learning rate η, Criteria or value ϵ
Result: Total Error E_{total}
while $E_{total} < \epsilon$ *or not_convergence* **do**
  for $\forall w$ **do**
    $\Delta w = -\frac{\partial E_{total}}{\partial}$ (equation: 4.18)
    $w_{new} \leftarrow w_{old}\ \eta \Delta w + \ldots$ (equation 4.12)
    Calculate *output* and E_{total}
  end
end

We can then redefine the gradient computation from Equation 4.18.

$$\Delta p \times w_{ti} = -\frac{\delta E_p}{\delta w_{ti}} \tag{4.20}$$

As already defined in Equation 4.18,

$$-\frac{\partial E_{total}}{w} = -\sum_p \frac{\partial E_p}{\partial w} \stackrel{def}{=} \sum_p \Delta_p w \tag{4.21}$$

Before applying the chain rule for Equation 4.20, let us define *net* (Equation: 4.22) and *out* (Equation: 4.23). The *net* equation refers to the scalar product (or dot product) between the weights and input vectors[3]. The f in the *out* refers to the activation function.

$$net = x^T w \tag{4.22}$$

$$out = f_\sigma(net) = f_\sigma(x^t w) \tag{4.23}$$

Let us now apply the chain rule for equation 4.20:

$$-\frac{\delta E_p}{\delta w_{ti}} = -\frac{\delta E_p}{\delta net_t} \times \frac{net_t}{\delta w_{ti}} \tag{4.24}$$

Let us proceed by splitting the right side of the equations into two parts. First the second part of the equation:

$$\frac{\delta net_t}{\delta w_{ti}} = \frac{\delta \sum_j w_{tj} out_j}{\delta w_{ti}} \tag{4.25}$$

Recalling the Equations for *net* 4.22 and 4.23 with respect to generic unit t, with a as an generic index for the summation internally.

[3] `https://www.mathsisfun.com/algebra/vectors-dot-product.html`

$$\begin{cases} net_t = & \sum_a w_{ta} out_a \\ out_t = & f_t(net_t) \end{cases} \tag{4.26}$$

Except for the situation where $j = i$, all of the components of 4.25 equals zero. The equation signifies that we have an input i from our generic unit out_i to the generic unit t. As a result, we achieve:

$$\frac{\delta \sum_j w_{tj} out_j}{\delta w_{ti}} = 0 + \ldots + \frac{\delta \sum_j w_{tj} out_j}{\delta w_{ti}} + \ldots + = out_i \tag{4.27}$$

For the first part of Equation 4.24, let us dedicate a new variable and apply the chain rule (4.15).

$$\delta t = -\frac{\delta E_p}{\delta net_t} = -\frac{\delta E_p}{\delta out_t} \times \frac{\delta out_t}{\delta net_t} = -\frac{\delta E_p}{\delta out_t} \times f_t'(net_t) \tag{4.28}$$

Summarizing the redefinition of equation 4.20.

$$\Delta p \times w_{ti} = \delta t \times out_i \tag{4.29}$$

We now have to solve the last portion of equation 4.30:

$$-\frac{\delta E_p}{\delta out_t} \tag{4.30}$$

We have to separate the situations for where our generic unit t is located.

1. $t \rightarrow$ output unit
2. $t \rightarrow$ input unit

Figure 4.12 illustrates a small proportion of the MLP, for a better insight.

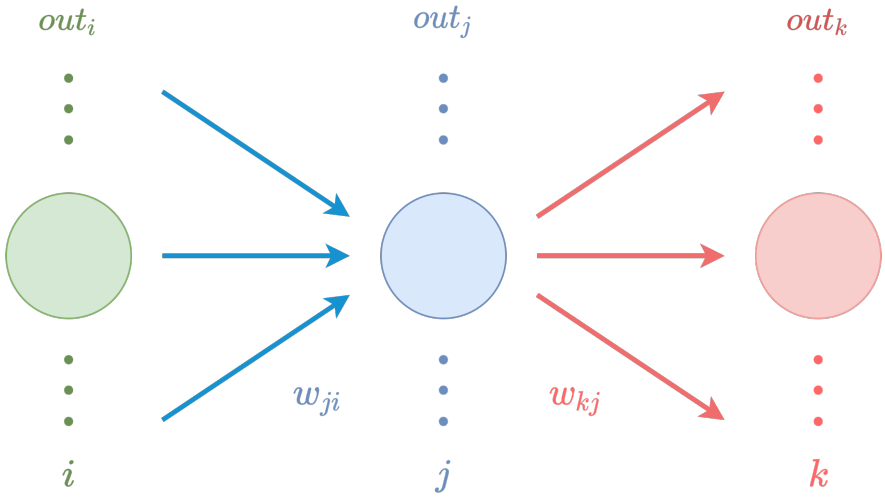


Figure 4.12: A small proportion of the MLP

Situation one: t is an output unit (k):

$$E_p = \frac{1}{2}\sum_k (d_k - out_k)^2 \tag{4.31}$$

We do have to notice the difference between k and k', where k refers to a constant related to *out*, while derivation, and k' refers to the summation internally.

$$-\frac{\delta f(w)^n}{\delta w} = n \times f(w)^{n-1} \times \frac{\delta f(w)}{\delta w} \tag{4.32}$$

We combine Equation 4.31 and 4.32:

$$-\frac{\delta E_p}{\delta out_k} = -\frac{\frac{1}{2} \times \delta \sum_{k'} (d_{k'} - out_{k'})^2}{\delta out_k}$$

$$= -\frac{1}{2} \times 2 \times (d_k - out_k) \times \frac{\delta (d_k - out_k)}{\delta out_k}$$

$$= -(d_k - out_k) \times (0 - 1) = (d_k - out_k)$$

Finally we get:

$$-\frac{\delta E_p}{\delta out_k} = (d_k - out_k) \tag{4.33}$$

We can obtain the derivative of the output unit k, by combining Equation 4.28 and 4.33:

$$\delta k = -\frac{\delta E_p}{\delta out_k} \times f'_k(net_k) = (d_k - out_k) \times f'_k(net_k) \tag{4.34}$$

Since it contains the discrepancy between the generated output and the desired target, this is an error signal (gradient error).

Situation two: t is an hidden unit (j): Because the error signal is not observable, it is difficult for the model to allocate the appropriate credit to hidden units. This problem is often referred to as the Credit Assignment Problem - CAP. The backpropagation algorithm was developed for this problem since we do not have the essential data to update weights based on the error for these particular hidden units[70].

We have that $t = j$, and now let us solve for:

$$-\frac{\delta E_p}{\delta out_j} \tag{4.35}$$

The fundamental element of the backpropagation method is to incorporate all of the hidden units' activities across all of the output units, where we sum them together. Equation 4.36, represents the variations that occurs on the Error E, with units k in mind. There are two variables which is dependent on out_j: out_k and net_k, which is the reason we use $\sum_k$.

$$-\frac{\delta E_p}{\delta out_j} = \sum_k -\frac{\delta E_p}{\delta net_k} \times \frac{\delta net_k}{\delta out_j} \tag{4.36}$$

We have now decomposed the *net* of output layer (k) with error in consideration, and the net of k with output units j. To now solve the equation, notice that the first part is already solved by us in Equation 4.34, therefore we can focus on the second part of Equation 4.36.

$$\frac{\delta net_k}{\delta out_j} = \frac{\delta \sum_{k'} w_{kk'} out_{k'}}{\delta out_j} = 0 + \ldots + \frac{\delta out_j}{\delta out_j} + \ldots + 0 = w_{kj} \tag{4.37}$$

Every partial derivatives are zero, except when $k' = j$. We can now combine together Equation 4.34 and 4.37, where we can define the solution for δj as:

$$-\frac{\delta E_p}{\delta out_j} \sum_k \delta k \times w_{kj} \tag{4.38}$$

As a result, we may compose the error signal for the hidden units in *Situation two* as:

$$\delta j = \sum_k (\delta k \times w_{kj}) \times f_j'(net_j) \tag{4.39}$$

The inclusion of the weight w_{kj} in 4.39 signifies that the k is being sent backward from the output layer to the hidden layer, and this weight is the link between them. Meaning we are propagating the error signal backward - Backpropagation.

We have now solved the partial derivatives for the output (4.34) and hidden units (4.39), whereas we can now upgrade the weight. Based on which situation - either Situation one with t as an output unit or Situation 2 with t as a hidden unit, we can upgrade the weight with the delta rule (Equation: 4.20), where Δt takes corresponding form based on the listed situations (Equation: 4.34 or 4.39)

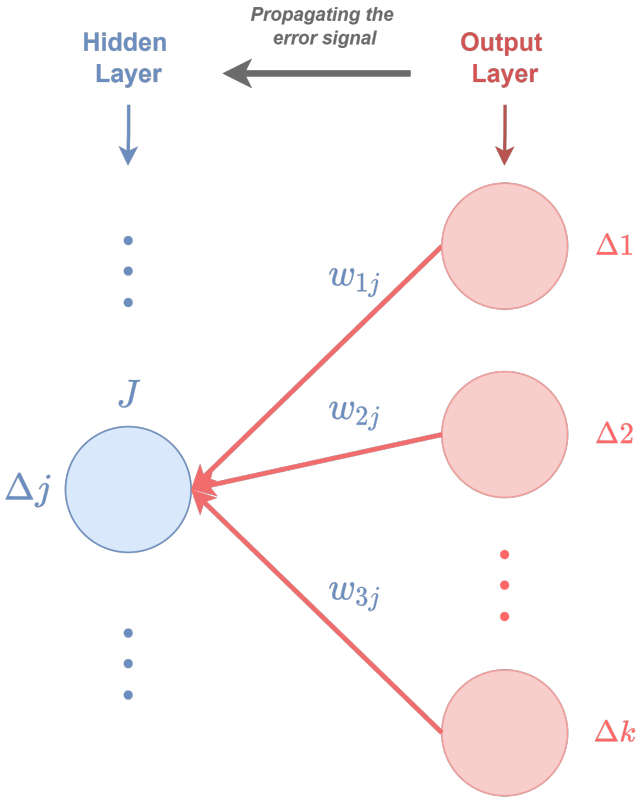


Figure 4.13: Backpropagation of Delta values from output layer to hidden layer

The contribution of $\Delta 1, \ldots, \Delta k$ at each output unit k in Figure 4.13, which is then totaled, weighted with the relevant connection weight w_{kj}, and sent to the hidden unit j. Naturally, this procedure applies to each hidden unit j. In some ways, the related weight w_{kj} may be considered a measure of how much the unit k is to account for the error signal. In this situation, the network's overall liability for the entire error will be spread according to w_{kj} weights[70].

4.1.7 Vanishing- and Exploding Gradients problem

We will introduce some of the challenges related to activation functions and backpropagation: Vanishing- and Exploding Gradients.

Gradient

The weighed derivative of the error function is referred to as a gradient. We utilize the gradient to update the weights in a neural network to minimize the error (Loss function) throughout backpropagation when traversing from the output layer to the input layer[82].

Vanishing Gradient Problem

When we travel backward with each layer during backpropagation, the derivative or slope gets less and smaller until it vanishes. When the weight update is very little or exponentially small, the training period is excessively long, and in the worst situation, the neural network training is stopped.

A Vanishing Gradient problem is often related to two main activation functions: Sigmoid (Section: 4.1.8) and TanH (Section: 4.1.8). This phenomenon is caused by the derivative of the functions operates in values between 0 to 0.25 (TanH) and 0 to 1 (Sigmoid). The new weight values are close to the old weight values, and the modified weight values are insignificant. As a result, the gradient descent algorithm never reaches the best solution[82].

Exploding Gradient Problem

During backpropagation, an exploding gradient happens when the derivatives or slope become more significant as we move backward with each layer. This phenomenon is the polar opposite of the vanishing gradient problem.

This issue arises due to weights rather than the activation function. The derivatives are likewise high because of the significant weight values, causing the new weight to differ significantly from the previous weight and the gradient never to converge. Exploding gradients can lead to an unstable network that cannot learn from the training data and cannot be updated at worst, resulting in `NaN` weight values that cannot be updated. As a result, it is possible that the system will oscillate about minima, never reaching a global minimum, as shown in Figure: 4.6).

The Vanishing and Exploding Gradients problem are illustrated in Figure 4.14

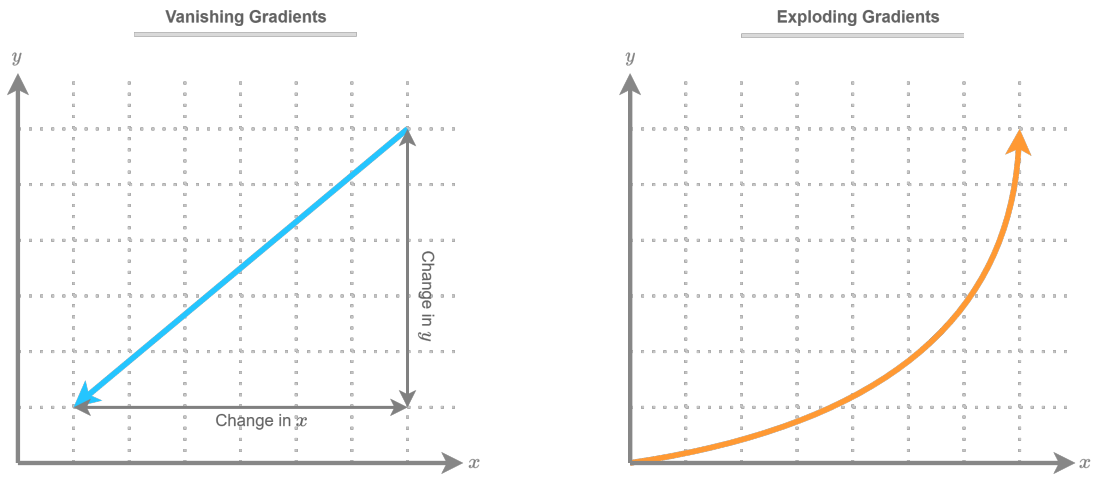


Figure 4.14: Vanishing and Exploding Gradients

4.1.8 Activation Functions

In a neural network, an activation function specifies how the weighted sum of the input is turned into an output from nodes in a layer. The activation function is often called a ”transfer function”, and a ”squashing function” if its output range is restricted[72], [81]. In addition, many activation functions are non-linear, which is referred to as ”nonlinearity” in a network architecture.

The activation function chosen has a significant influence on the neural network's capabilities and performance, and different activation functions may be utilized in other sections of the model.

Activation functions are usually differentiable, which means that for a given input value, the first-order derivative can be determined. This is necessary because neural networks are commonly learned using the backpropagation of error algorithm, which involves the derivative of prediction error to adjust the model's weights. In neural networks, there are several distinct types of activation functions, albeit possibly only a few are employed in practice for hidden and output layers. Figure 4.15 illustrates a generic activation function.

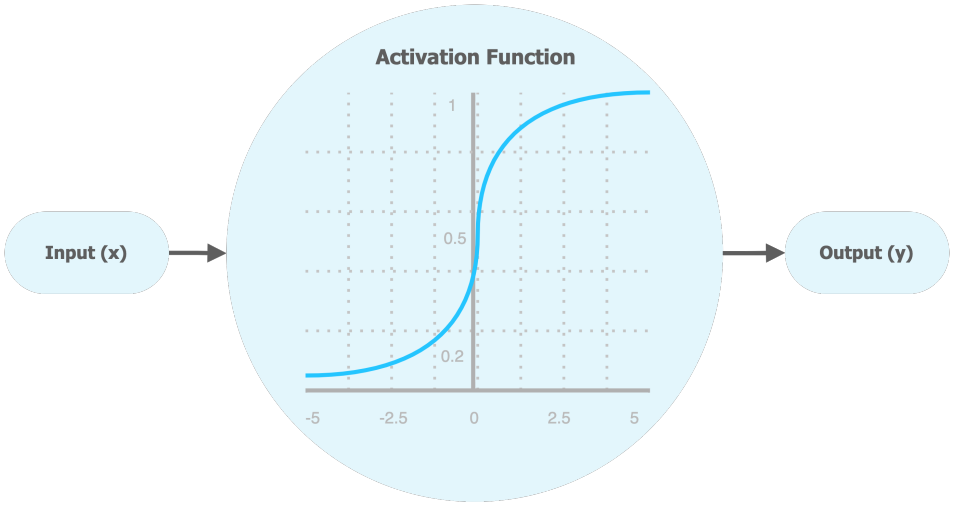


Figure 4.15: Activation Function

Linear Activation Function

Where the output is proportionate to the input, it represents linearity, which is known as a Linear function or Identity function. Mathematical representation of Linear activation function, shown in Equation 4.40. The Linear activation function is represented visually in Figure 4.16.

$$Linear(x) = x \tag{4.40}$$

Linear activation function has two fundamental drawbacks:

1. Backpropagation is not conceivable because the function's derivative is a constant and has no relationship to the input x.

2. If a Linear activation function is implemented, all layers of the neural network will collapse into one. The last layer of the neural network will always be a Linear function of the first layer, regardless of how many layers there are. Hence, a Linear function effectively reduces the neural network to a single layer.

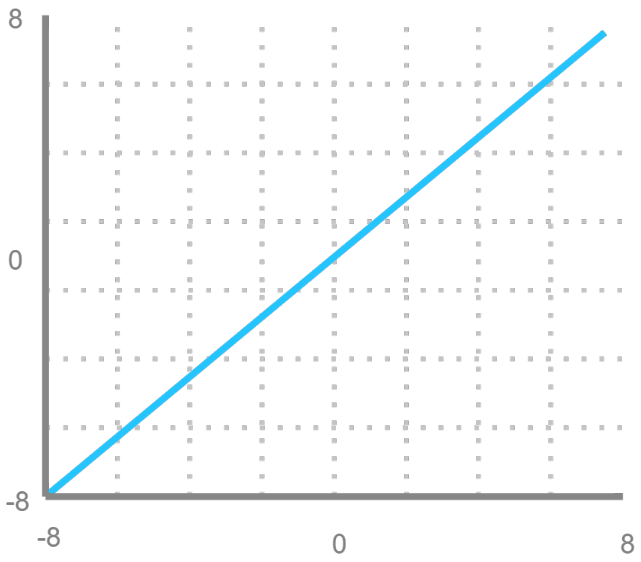


Figure 4.16: Linear Activation Function

Sigmoid (Logistic)

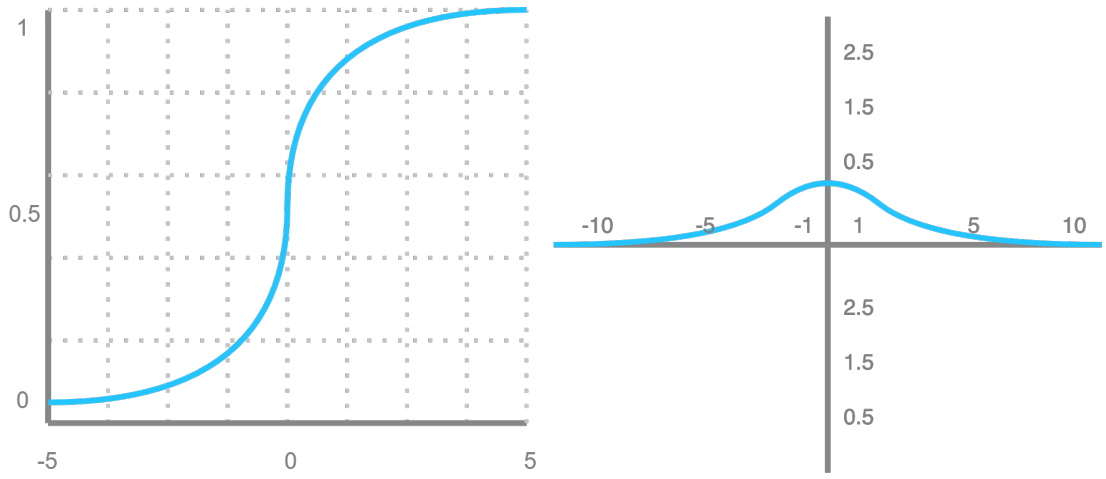
Studying how a neural network learns complex tasks requires understanding the Sigmoid function. The Sigmoid function has also been used as a core technology that led to other efficient and promising supervised learning solutions in deep learning systems.

The Sigmoid function is a particular logistic function indicated by the letters $\sigma(x)$ or $sig(x)$. It is provided by Equation 4.41

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4.41}$$

Because its domain and range (0, 1) are the set of all real numbers, the Sigmoid function is also known as a squashing function. As a result, the output is always between 0 and 1 if the function's input is either a substantial negative or positive value. The same rules apply for any number in the range of $-\infty$ to $+\infty$.

We can only learn linearly separable issues when we utilize a Linear activation function in a neural network. With just one hidden layer and a Sigmoid activation function in the



((a)) Sigmoid Activation Function

((b)) Sigmoid Derivative

Figure 4.17: Sigmoid Functions

hidden layer, the neural network can learn a non-linearly separable problem effectively. The Sigmoid function may be utilized in neural networks to learn complicated decision functions since it provides non-linear boundaries when using a non-linear function.

As illustrated in Figure 4.17(a), the greater the input (more positive), the closer the output value is to 1.0, and the smaller the input (more negative), the closer the output is to 0.0.

The following are some of the reasons why the Sigmoid/logistic activation function is so popular:

1. It is widely employed in models where the output is a probability prediction. Because probability only exists in the range of 0 to 1, the Sigmoid is the best choice due to its range.

2. The function is differentiable and has a smooth gradient, excluding output value jumps.

The main disadvantage of the Sigmoid function lies in the derivative, which is mathematically represented in Equation 4.42. The gradient values are only relevant for the range -5 to 5, as seen in Figure 4.17(b), and the graph becomes considerably flat in other regions.

$$\sigma'(x) = \sigma(x) \times (1 - \sigma(x)). \tag{4.42}$$

It suggests that the function has very modest gradients for values more than 5 or less than -5. The network stops learning as the gradient value approaches zero, resulting in the Vanishing gradient problem. The Logistic function's output is asymmetric near zero. As a result, each neuron's output will have the same sign, making neural network training

more challenging and unreliable. The Sigmoid activation function can be implemented in Python, as shown in Listing 4.5.

```
import numpy as np

# Sigmoid Activation Function
def sigmoid(x):
  return 1/(1+np.exp(-x))

# Derivative of Sigmoid
def sigmoid_derivative(x):
  return sigmoid(x) * (1- sigmoid(x))
```

Listing 4.5: Sigmoid implementation in Python

Hyperbolic Tangent (Tanh)

With a difference in an output range of -1 to 1, the Tanh function is remarkably similar to the Sigmoid (logistic) activation function and has a similar S-shape, shown in Figure 4.18(a)). The greater the input (more positive), the closer the output is to 1.0, and the lower the input (more negative), the closer the output is to -1.0. The Activation function is represented mathematically in Equation 4.43

$$Tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4.43}$$

The Tanh Activation function offers some advantages:

1. We may translate the output values as strongly negative, neutral, or very positive Since the Tanh function's output is centered around Zero.

2. It is widely used in hidden layers of neural networks since its values range from -1 to 1. Consequently, the average of the hidden layer is 0 or very similar to it. It assists with data centering and makes understanding the next layer easier.

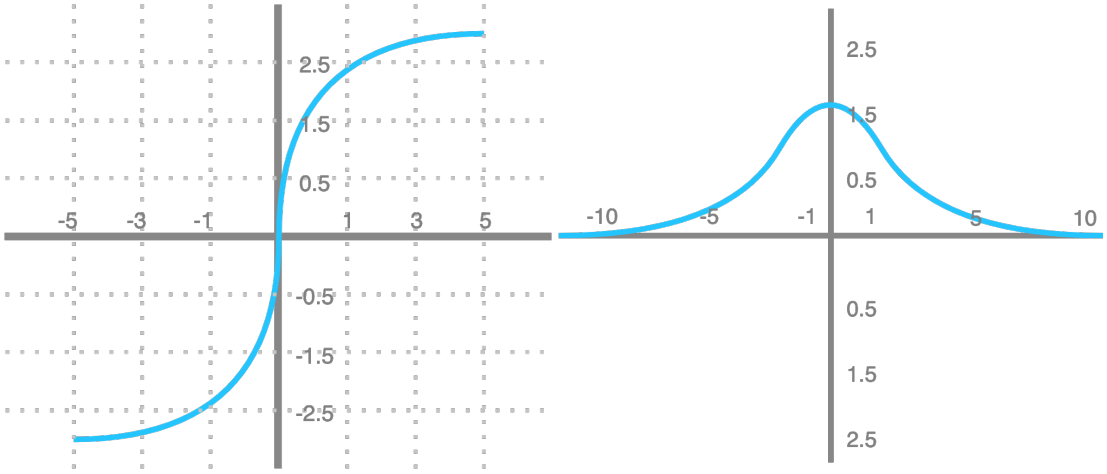
Like the Sigmoid activation function, from Section 4.1.8, it has the difficulty of vanishing gradients. In addition, the Tanh function has a substantially steeper gradient (Figure: 4.18(b)) than the Sigmoid function. The Tanh activation can be implemented in Python as shown in Listing 4.6[83]

```
import numpy as np

# Hyperbolic Tangent (tanh) Activation Function
def tanh(x):
  return (np.exp(x) - np.exp(-x))/(np.exp(x) + np.exp(-x))

# tanh derivative
def tanh_derivative(x):
  return 1 - tanh(x) * tanh(x)
```

Listing 4.6: Hyperbolic Tangent (Tanh) implementation in Python



((a)) Tanh Activation Function

((b)) Tanh Derivative

Figure 4.18: Tanh Functions

Limitations of Sigmoid and Tanh

The Sigmoid and Tanh functions are both saturated, which is a common issue. This means that for Tanh and Sigmoid, high values bounce to 1.0, and low values bounce to -1 or 0. Furthermore, the functions are only responsive to changes in their input around the mid-region, around 0.5 for Sigmoid and 0.0 for Tanh.

Regardless of whether the summed activation from the node provided as input includes meaningful information or not, the function has limited sensitivity and saturation. When the learning algorithm becomes saturated, it becomes problematic to continue adjusting the weights to enhance the model's performance.

Rectified Linear Unit (ReLU) Activation Function

ReLU, shown in Figure 4.19, is an abbreviation for Rectified Linear Unit. ReLU has a derivative function and enables backpropagation while being computationally efficient, despite its seeming to be a Linear function. The fundamental limitation is that the ReLU function does not simultaneously fire all neurons. Only if the result of the linear transformation is less than 0 will the neurons be deactivated.

Mathematical representation of Rectified Linear Unit (ReLU) in Equation 4.44

$$ReLU(x) = max(0, x) \tag{4.44}$$

Rectified Linear Unit offers some advantages:

1. Compared to the Tanh and Sigmoid activation functions, the ReLU function is considerably more computationally efficient. This is because only a small number of neurons are active.

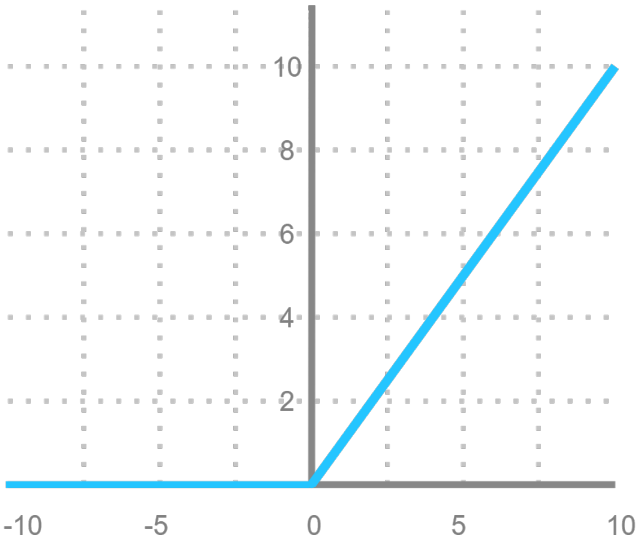


Figure 4.19: Rectified Liner Unit Activation Function

2. Due to its linear, non-saturating characteristic, the Gradient Descent converges fast towards the global minimum of the Loss function

The Dying RelU problem is one of the disadvantages of this function. The gradient value is 0 on the negative partition of the graph, in Figure: 4.20. As a result, some neurons' weights and biases are not updated throughout the backpropagation process. This can result in neurons that are never fired. Python implementation for ReLU is represented in Listing 4.7[84].

```
import numpy as np

# Rectified Linear Unit (ReLU)
def ReLU(x):
  data = [max(0,value) for value in x]
  return np.array(data, dtype=float)

# Derivative for ReLU
def ReLU_derivative(x):
  data = [1 if value>0 else 0 for value in x]
  return np.array(data, dtype=float)
```

Listing 4.7: Rectified Linear Unit (ReLU) implementation in Python

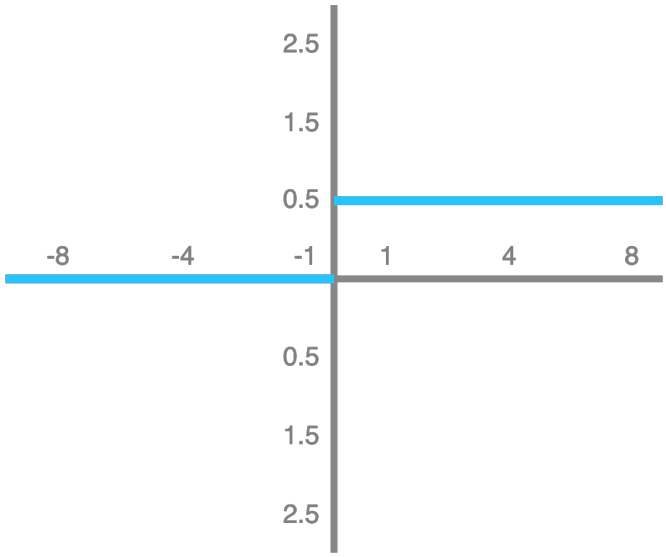


Figure 4.20: The Dying RelU Problem

Leaky Rectified Linear Unit (LeakyReLU)

Leaky Rectified Linear Unit (LeakyReLU) typically has an alpha (α) between 0.1 and 0.3. Therefore, the LeakyReLU activation function is widely utilized. However, it has certain disadvantages compared to the ELU and some advantages when compared to ReLU.

Instead of a smooth slope, it features a small one for negative numbers, as shown in Figure 4.21. Furthermore, the coefficient for this particular slope is calculated prior to training rather than learned during training. This activation function is often used in applications involving sparse gradients[4], such as training Generative Adversarial Networks (GAN).

The Equation for LeakyReLU 4.45 addresses two significant problems: the dying ReLU problem since the values of the gradients do not stop at the value zero, and the vanishing Gradient problem. Correspondingly, the derivative of LeakyReLU is mathematically represented in Equation 4.46. Finally, the Python implementation for LeakyReLU is presented in Listing 4.8.

$$LReLU(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases} \tag{4.45}$$

$$LReLU'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha & \text{if } x \leq 0 \end{cases} \tag{4.46}$$

[4]Sparse gradients indicate that a network is not getting enough information to tweak its weights. A good example involving sparse gradients is the vanishing gradient problem. In other words, repeatedly multiplying small numbers yields progressively more minor results. As a result, the little values that arise as the gradient are insufficient for the network to modify its weights.

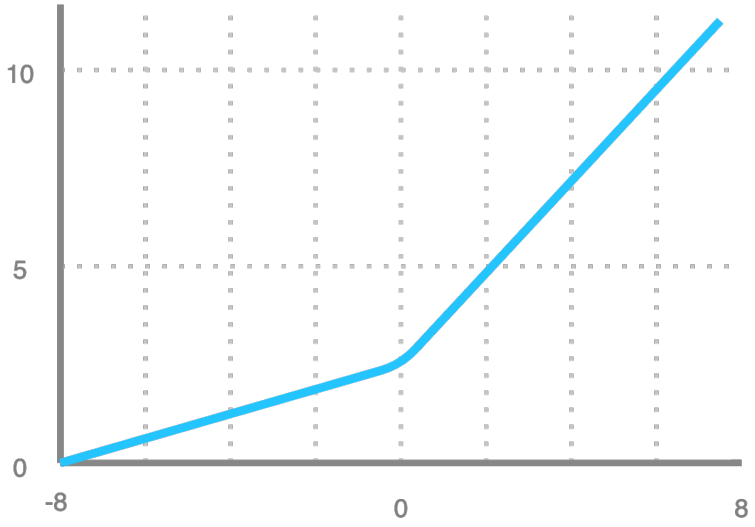


Figure 4.21: LeakyReLU Activation Function

```python
import numpy as np

# Leaky Rectified Linear Unit (leaky ReLU) Activation Function
def leakyReLU(x):
  data = [max(0.05*value,value) for value in x]
  return np.array(data, dtype=float)

# Derivative for leaky ReLU
def leakyReLU_derivative(x):
  data = [1 if value>0 else 0.05 for value in x]
  return np.array(data, dtype=float)
```

Listing 4.8: LeakyReLU implementation in Python

Parametric ReLU (PReLU)

Parametric ReLU[85], shown in Figure 4.22, is a variation of the ReLU, which seeks to overcome the challenge of the axis becoming zero on the left half side. As an input a, the function returns the slope of the negative element of the function, which is used to determine the most acceptable value of a during backpropagation. The PreLU activation function can be mathematically expressed as Equation 4.47, where a is the parameter for the slope (negative values). If the LeakyReLU activation function fails to solve the issue of dying neurons and essential information is not passed on to the next layer, the Parametric ReLU is utilized. Depending on the slope parameter a, the performance of the functions depends on the various problems, which might be a drawback. The Python implementation for PreLU is presented in Listing 4.9

$$PReLU(x) = max(ax, x) \tag{4.47}$$

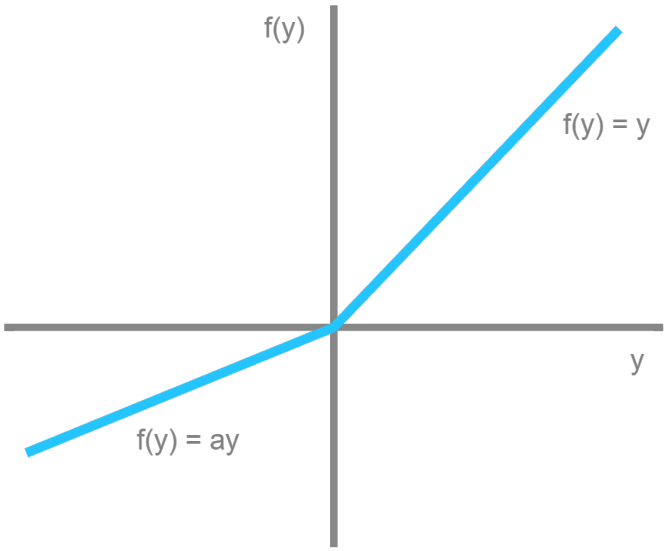


Figure 4.22: Parametric ReLU Activation Function

```python
import tensorflow as tf
import torch.nn as nn

# Parametric Rectified Linear Unit - LeakyReLU
def parametric_relu(_x):
    alphas = tf.get_variable('alpha', _x.get_shape()[-1],
                             initializer=tf.constant_initializer(0.0),
                             dtype=tf.float32)
    pos = tf.nn.relu(_x)
    neg = alphas * (_x - abs(_x)) * 0.5

    return pos + neg
```

Listing 4.9: Parametric ReLU implementation in Python

Exponential Linear Units (ELU)

The ELU (Exponential Linear Unit)[86], shown in Figure 4.23, is a variation of RelU that changes the slope of the function's negative portion. Unlike other activation functions, ELU contains an additional alpha constant that must be positive.

With the exception of negative inputs, ELU is highly similar to ReLU. Furthermore, the ELU activation function provides a non-zero gradient for $x = 0$, eliminating the problem of dying RelU (Neurons dying). Furthermore, this function is smooth throughout, including at $x = 0$. This accelerates the learning process.

The ELU activation function can be presented mathematically as in Equation 4.48.

$$ELU(x) = \begin{cases} x & \text{for } x \geq 0 \\ \alpha(e^x - 1) & \text{for } x < 0 \end{cases} \tag{4.48}$$

The main disadvantage of ELU is that it takes longer to calculate than ReLU and its variations. This is due to ELU's usage of an exponential function that is computationally demanding. During training, however, the quicker convergence rate compensates for this. As a result, an ELU network will be slower throughout testing than a ReLU neural network.

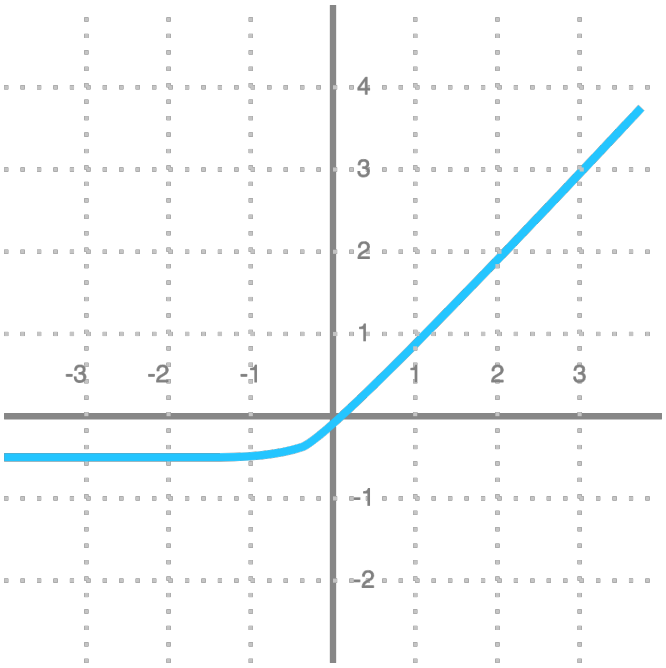


Figure 4.23: Exponential Linear Units (ELU) Activation Function

Scaled Exponential Linear Unit (SELU)

The Scaled Exponential Linear Unit (SELU), shown in Figure 4.24, handles normalization internally, first presented in self-normalizing networks[87]. Self-normalizing Neural Networks (SNNs), per definition, maintain the zero-mean and unit-variance activations automatically per neuron[5]. SELU allows for this standardization by altering the mean and variance, implying that each layer keeps the mean and variance from the last layer. Furthermore, since ReLU (Section 4.1.8) cannot produce negative values, SELU has a direct advantage as it can handle both positive and negative values to shift the mean. To alter the variance, gradients can be utilized. Raising the activation function requires an area with a gradient more significant than one[72].

[5] https://dl.acm.org/doi/10.5555/3294771.3294864

To summarize, SELUs have a great self-normalization property while eliminating the vanishing gradients problem[88]. The SELU activation function has predefined parameters: $\alpha = 1.6732$ and $\lambda = 1.0507$. Mathematically the activation function is represented in Equation 4.49, and the Python implementation for SELU is presented in Listing 4.10.

$$SELU(x) = \begin{cases} \lambda x & \text{if } x \geq 0 \\ \lambda\alpha(e^x - 1) & \text{if } x < 0 \end{cases} \tag{4.49}$$

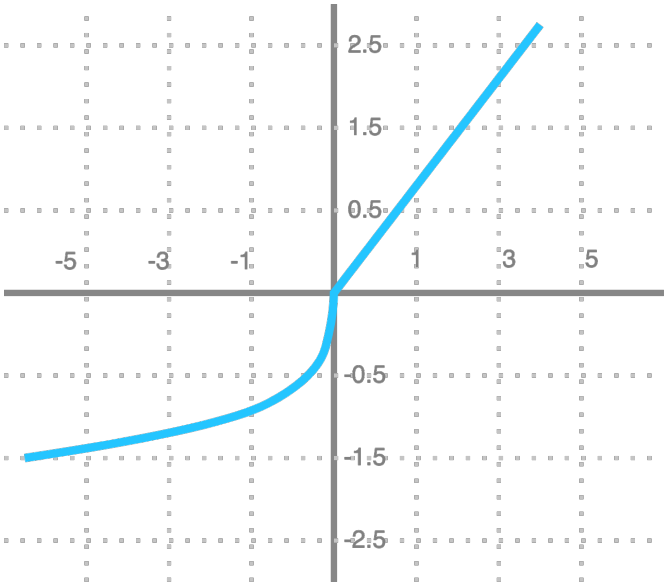


Figure 4.24: SELU Activation Function

```python
import numpy as np

# Scaled Exponential Linear Units
def SELU(x, lambdaa = 1.0507, alpha = 1.6732):
    if x >= 0:
        return lambdaa * x
    else:
        return lambdaa * alpha * (np.exp(x) - 1)
```

Listing 4.10: Scaled Exponential Liner Units implementation in Python

Gaussian Error Linear Unit (GELU)

GELU, known as Gaussian Error Linear Unit[89], is an activation function. GELU can be described as $x\phi(x)$, where the standard Gaussian cumulative distribute function is involved as $\phi(x)$.

GELU, unlike the variations of ReLU, assigns weights to the inputs based on their values instead of their sign when thresholding. GELU and ReLU are distinguished because, unlike ReLu, GeLU could be both negative and positive. Compared to ReLU and ELU, it is observed that GELU activation enhances performance across the board in computer vision, natural language processing, and speech applications..

We define the Gaussian Error Linear Unit (GELU) in Equation 4.50 since the cumulative distribution function of a Gaussian is frequently estimated with the error function. Further, we may approximate GELU with Equation 4.51. The GELU activation can be implemented in Python, as shown in Listing 4.11[90]. Furthermore, Figure, 4.25 represents a comparison of ELU, ReLU and GELU.

$$\text{GELU}\,(x) = xP\,(X \leq x) = x\Phi\,(x) = x \cdot \frac{1}{2}\left[1 + \text{erf}(x/\sqrt{2})\right] \qquad \text{if } X \sim \mathcal{N}(0,1) \quad (4.50)$$

$$GELU(x) = 0.5\left(1 + Tanh\left[\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)\right]\right) \quad (4.51)$$

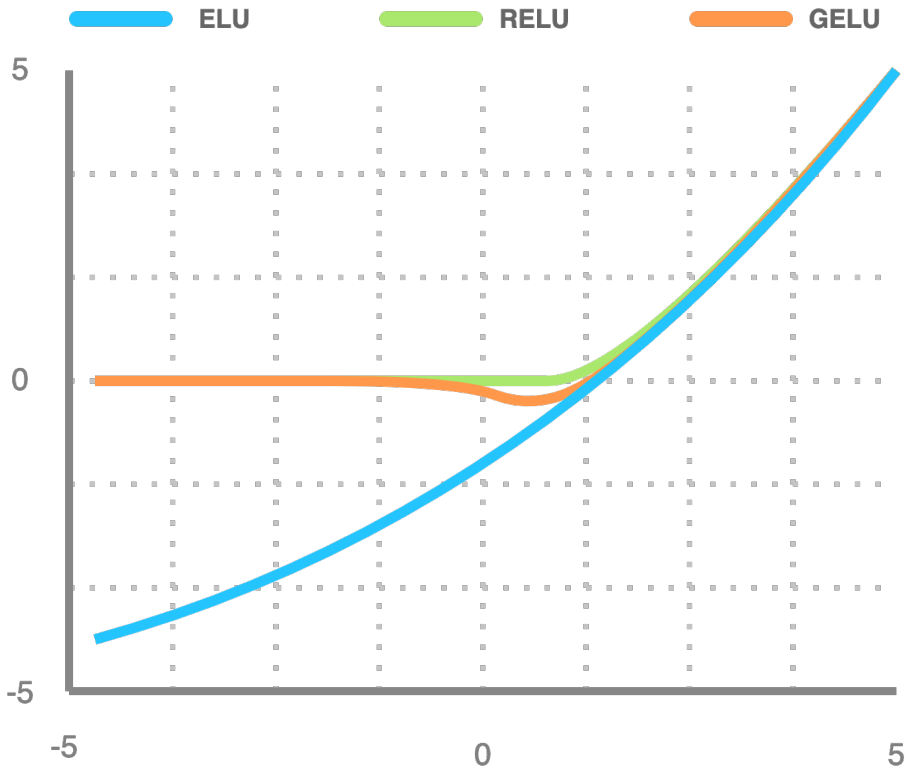


Figure 4.25: GELU Activation Function

```python
from scipy.stats import norm

def gelu(x):
    return x*norm.cdf(x)
def gelu_approx(x):
    return .5 * x * 1(1 + tanh(np.sqrt(2/np.pi)*(x+0.044715 * x**3))
```

Listing 4.11: GELU Activation Function Implementation in Python

Softmax

The Softmax activation function, shown in Figure 4.26, is a well-known function in ML, particularly in deep learning. In ANN, the Softmax is frequently inserted in the final layer. Softmax is a function that specializes in converting integers into probabilities. As a result, the Softmax output produces a vector v, with probabilities of conceivable prediction. The Softmax activation function can be defined mathematically in Equation 4.52[91], where the notations are presented in Table 4.4. The Python implementation for Softmax is presented in Listing 4.12.

Table 4.4: Softmax Notations

Notation	Explanation
y	The input vector for the Softmax function, which consists of n classes (possible outcomes)
y_i	Refers to the i-th element from the input vector y. The value can range from $-\infty$ $+\infty$
e^{y_i}	e refers to the standard exponential function applied to y_i. An outcome is a small number (Close to 0, but never 0). Whereas a large number if $y_i < 0$. We can define $e \approx 2.718$, the Euler's number.
$\sum_{j_i}^{n} e^{y_i}$	Refers to a normalization term. It assures that output vector $SoftMax(y)_i$ accumulates to 1 for the i-th class, and that each value is between 0 and 1, resulting in an acceptable probability distribution
n	The number of total classes, possible outcomes.

$$Softmax(y)_i = \frac{e^{(y_i)}}{\sum_{j_i}^{n} e^{(y_j)}} \tag{4.52}$$

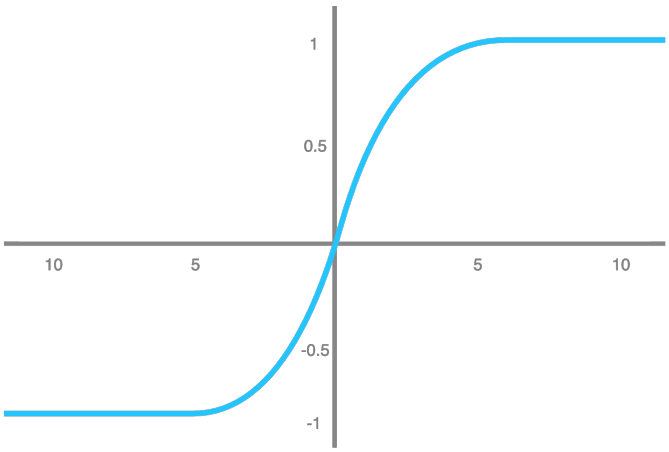


Figure 4.26: Softmax Activation Function

```python
from math import exp

# Softmax Activation Function
def SoftMax(inputVector):
    # Calculating the exponent for each element in the input vector
    exponents = [exp(j) for j in inputVector]

    # Dividing the exponent of valuue by the sum of the exponents.
    # Round to 3 decimals.
    p = [round(exp(i)/sum(exponents), 3) for i in inputVector]
    return p
```

Listing 4.12: Softmax implementation in Python

Binary Step

A threshold value determines whether a neuron should be fired or not in a Binary Step function, shown in Figure 4.27. The input to the activation function is compared to a threshold; if it is higher, the neuron is activated; if it is lower, the neuron is deactivated, and its output is not sent on to the next hidden layer. The Binary Step function can be represented mathematically in Equation 4.53, where the Python implementation is shown in Listing 4.13[92].

$$Binary(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases} \tag{4.53}$$

The following are some of the drawbacks of the Binary Step function:

1. It cannot offer multi-value outputs. Hence it cannot be utilized to solve Multi-Class Classification tasks.

2. The step function's gradient is zero, making the backpropagation procedure difficult.

```python
# Binary Step Activation Function
def Binary_step(x):
    if x < 0:
        return 0
    else:
        return 1
```

Listing 4.13: Binary Step function implementation in Python

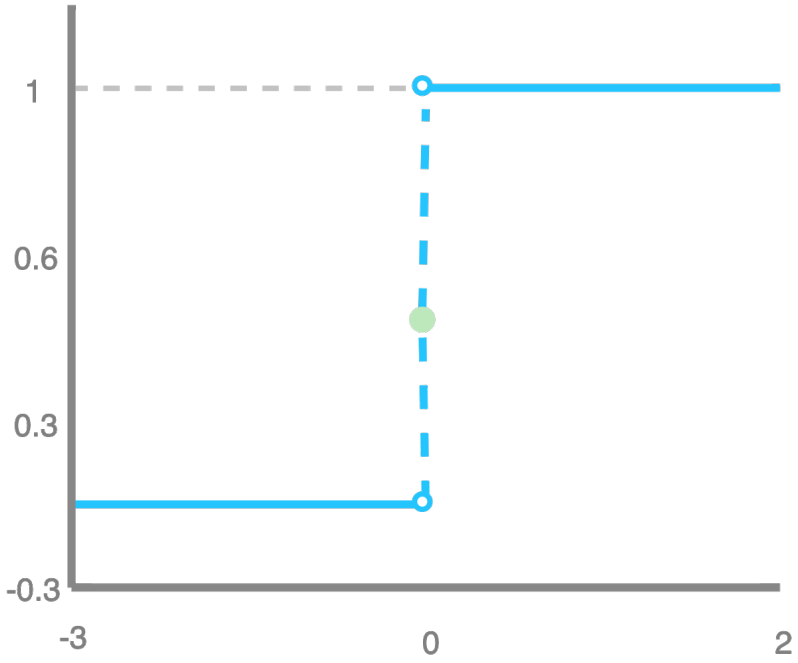


Figure 4.27: Binary Step Activation Function

4.2 Logistic Regression (LR)

Joseph Berkson developed Logistic Regression (LR) in 1944[93]. LR is a statistical model that predicts an event's probability out of two choices. This section will explain how LR works, predicts, and prepares the data for the algorithm.

4.2.1 Logistic Function

Logistic Regression is based on the Logistic function algorithm, which also can be called the Sigmoid function. The Logistic function was developed for ecology to show the growth of the population. The S-shaped function maxes out at a relatively quick pace. The function can take any number and map it into a value between 1 and 0. However, it will never be exactly one or zero [42]. The Sigmoid function can be referred to in Section 4.1.8, with Figure 4.17 representing the S-curve. Let us restate the Equation from Section 4.1.8 as Equation 4.54, where e is Euler's number and x is any given value from the input.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4.54}$$

4.2.2 Linear Regression

The x is the independent input variable in Linear Regression, and y is the dependent output variable. Figure 4.28 shows the separation line made by Equation 4.55, independent and dependent variable. Linear regression finds the relationship between output and input variables and then draws a line that maps the input data to the output data. The line which is drawn represents the relationship between the independent variables. It does this by avoiding outliers and covering as many input variables as possible.

$$y = a_0 + a_1 \times x + e \tag{4.55}$$

4.2.3 Logistic Regression

The equation for LR is similar to the Logistic function. LR takes an input value (x) and combines it linearly with weights (β) to predict the output (y). The difference between logistic function and LR is the output. Instead of getting a number between 0 and 1, the LR gets a binary output (1 or 0). LR can be mathematically represented with Equation 4.56:

$$y = \left(\frac{e^{(b0 \times b1 \times x)}}{1 + e^{(b0 + b1 \times x)}} \right) \tag{4.56}$$

Here y will be the output it predicts, $b1$ is the coefficient from the output of x, and $b0$ is the bias. Every column in the training data has a coefficient that it learns. This value is a significant value to store in the memory.

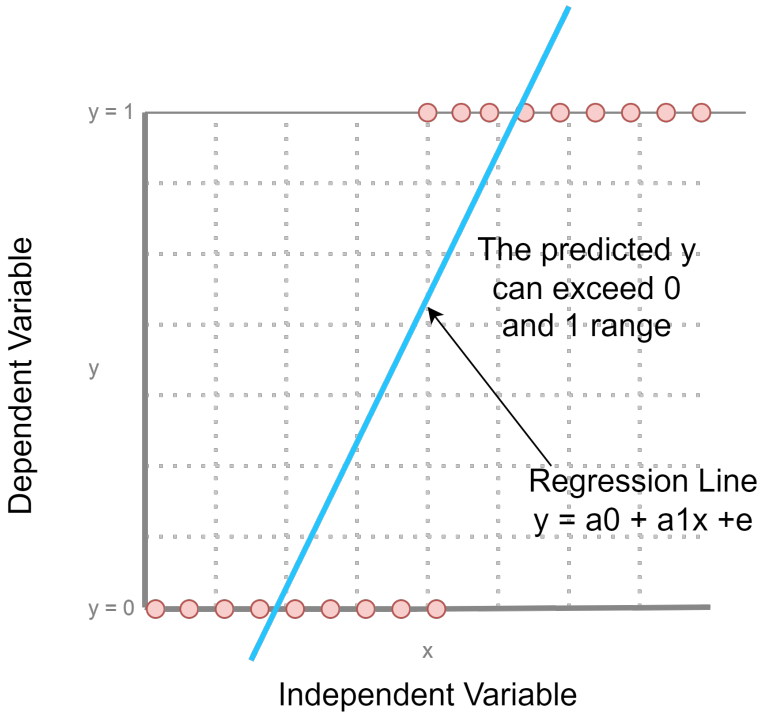


Figure 4.28: Linear Regression

4.2.4 How Logistic Regression Predicts the Probabilities

LR finds the probability of the first class. The first class can be represented with Equation 4.57. This is the probability that the input X belongs to the class $Y = 1$.

$$P(X) = P(Y = 1|X) \tag{4.57}$$

To make predictions, they need to be transformed by the Logistic function. This makes it so the predictions are no longer arranged as a linear combination. It is easier to understand Linear Regression since it is a linear model. The model can be stated with Equation 4.58.

$$P(X) = \frac{e^{(b0+b1\times X)}}{1 + e^{(b0+b1\times X)}} \tag{4.58}$$

The Equation for Probability Model rewritten is shown in Equation 4.59

$$\frac{ln(p(X)}{1 - p(X))} = b0 + b1 \times X \tag{4.59}$$

When the right side is calculated, it will be linear again, and the left side will be the *log* of the probability from the first class. The left side ratio is the odds of the default

class. Calculating the odds takes the probability of an event happening divided by the probability of an event not happening. It can be represented with Equation 4.60

$$ln(odds) = b0 + b1 \times X \tag{4.60}$$

The left side is now the probit[6]. There are multiple ways to transform those equations. The transformation process is performed using the probit link function for the Linear Regression equation. From Equation 4.60, it can move the exponent to the right side and get the final Equation 4.61, which helps to understand the linear combination combined with the odds of the first class.

$$odds = e^{(b0+b1\times X)} \tag{4.61}$$

4.2.5 Logistic Regression Model

When finding the coefficient β value, it has to be estimated from the training data, and it is done by using a maximum-likelihood estimator. The maximum-likelihood estimator makes assumptions about how the data is distributed. It is a popular method used by other ML algorithms[7]. The better the coefficient value, the closer the prediction gets to the predicted class. So if the predicted class belongs to the default class, the number will be close to one, but if it belongs to the other class, it will be close to 0. When the maximum-likelihood method is used for LR, its primary goal is to find a coefficient value that reduces the error by the maximum amount in the predicted probability compared to the actual value.

4.2.6 How To Make Predictions With Logistic Regression

Predictions with LR are made by feeding numbers into the LR equation. For example, predicting if an animal is a dog or a cat based on its weight. The animal's weight is $20kg$. By defining it more formally, it will be $P(dog|weight = 20)$. Having given the coefficient $b0 = -15$ and $b1 = 0.7$, and feeding those values into the Equation for LR 4.56, results in $y = e^{(-15+0.7\times20)/(1+e^{(-15+0.7\times20)})}$. This will give an output of 0.735, meaning it is highly likely for the animal to be a dog since the dog is the primary class, and the value is close to 1. Looking at it as binary prediction, anything above 0.5 is a dog, and anything under is a cat. A representation of this can be shown in Figure 4.29.

[6]The probit model is a type of regression where the dependent variable can take only two values(binary outcome variables)

[7]`https://machinelearningmastery.com/logistic-regression-for-machine-learning/`

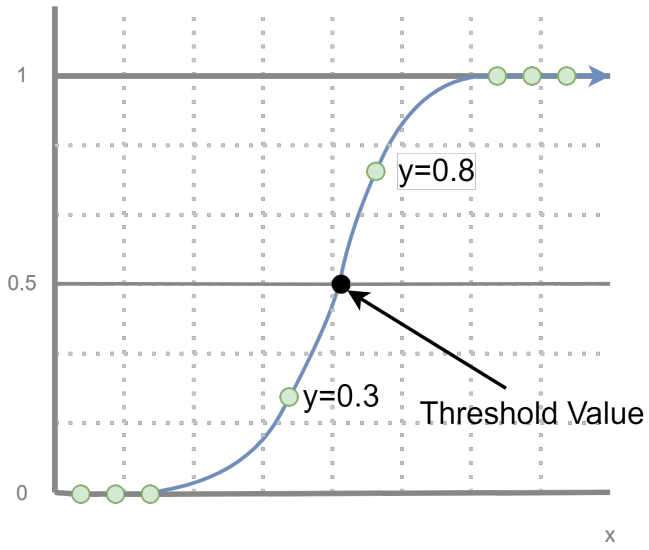


Figure 4.29: Linear Regression Threshold

4.2.7 Prepare The Data For Logistic Regression

As with most other ML algorithms, the preprocessing of the data is essential. Here Logistic Regression and Linear Regression have the same relationships and distribution, some of the same data preprocessing methods can be used for them both.

- **Remove Correlated Inputs:** If there are many inputs with high correlation, the model can overfit. This goes for Linear Regression and Logistic Regression. Calculating the input correlations by pair and removing the ones with too high correlation prevents the algorithm from overfitting.
- **Remove Noise:** Removing outliers and data that can misclassify from the training data will help LR perform better. Since LR assumes that there are no outliers or misclassified data.
- **Gaussian Distribution:** Transforming the input variables will make it easier to expose the linear relationship. By exposing the relationship, the model can become more accurate. Since LR is a linear algorithm, it assumes a linear relationship between the output variable and the input variable. A few methods that can be used to transform the input variables are *Box-Cox, log, and root*[42].
- **Binary Output Variable:** The output needs to be a binary output value since the algorithm will predict the probability of a sample belonging to the first class.
- **Fail To Converge:** If the data has many highly correlated inputs or is very sparse, the maximum-likelihood estimator that finds the coefficient value might fail to converge (refer to Section 4.1.5, Figure 4.7 for more information).

4.2.8 Hyperparameters

Let us define the essential hyperparameters for LR:

- **Penalty:** This parameter specifies which norm the penalty calculation is going to use. It can be either $l2$, $l1$, *elasticnet*(combination of $l1$ and $l2$), or *none*. Some penalties do not work with all the solvers.
- **C:** This parameter works as it does in SVM. It decided how strong the regularization would be. It must be a positive `float` number.
- **RandomState:** This parameter only works for a few solvers, and it defines the randomness of the splits. It must be an `integer` or a random state instance.
- **Solver:** Solvers are algorithms that LR will use for the optimization problem. There are five different solvers in the Scikit-learn library for LR: `newton-cg`, `lbfgs`, `sag`, `saga`, `liblinear`[8]. Each solver has different usecases: `liblinear` is better for small datasets than `sag` and `saga`, which are faster for bigger datasets. `Liblinear` cannot handle the multi-class problem as it is only used for one versus the rest problems.
- **max_iter:** This parameter defines the number of iterations it takes for the solvers to converge.

4.3 Support Vector Machines (SVM)

Support-Vector Machine (SVM) is a supervised ML algorithm used for regression and classification problems. Vladimir Vapnik developed this algorithm with his colleagues in 1992 at AT&T Bell laboratories[94]. SVM is a very robust prediction algorithm. It is based on the statistical learning framework proposed by Vapnik. When the training data is fed into the algorithm, it will categorize into two categories, making SVM a non-probabilistic binary linear classifier. SVM will map all new data points that it will predict into a space. The training data separates into categories to maximize the distance between the classes, where the latest data points will then fall into one of the classes. SVM can also perform non-linear classification by using a technique where it maps the data to a higher dimension. This is called the kernel trick, which will be explained in Section 4.3.6.

4.3.1 Hyper-Plane

Hyperplanes are the decision boundary for SVM. The hyperplane is the line that separates the different data points fed into the algorithm. The data points go into either side of the hyperplane and are then classified to their class. A hyperplane does not necessarily need to be a line. The data is mapped to a higher dimension depending on the input

[8] `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`

features. So if the input features are two, then the hyperplane is just a line, shown in Figure: 4.30. But if the input feature is three, it will be a plane, shown in Figure: 4.31. The dimensions of the hyperplane are equal to the dimensions of the data minus one. When the dimension of the input features exceeds three, it becomes hard to imagine how it looks. The Equation for Hyperplane is shown in 4.62.

$$b_0 + b_1 \times x_1 + b_2 \times x_2 ... + b_n \times x_n = 0 \tag{4.62}$$

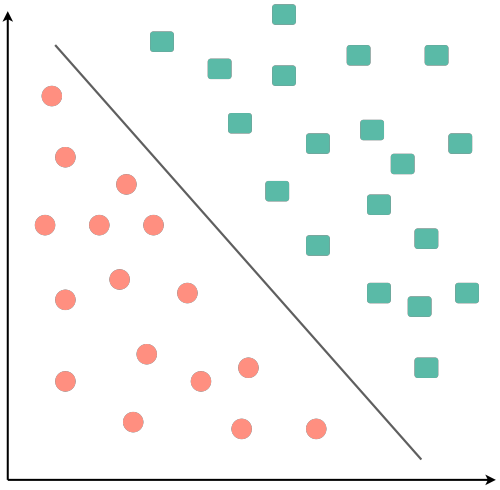


Figure 4.30: 2D Hyperplane

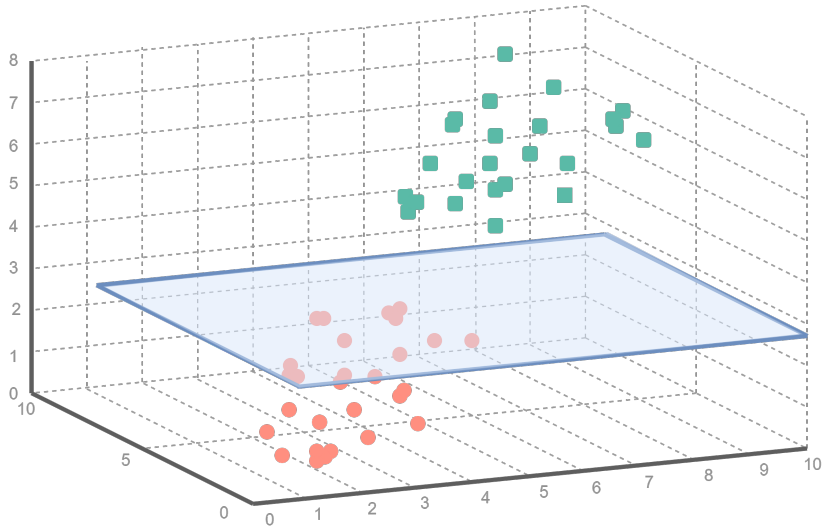


Figure 4.31: 3D Hyperplane

Linear Separable Cases

When SVM got has two input features, and the data points got a clearly separable case, as shown in Figure 4.30, then it needs to find the most optimal line to separate them. In the case of this example, there are many lines that can be made to separate the classes. As a result, the data points above the line will be fitting into Equation 4.64, and the data points below the line will fit into Equation 4.65, for the hyperplane 4.63.

$$b_0 + b_1 \times x_1 + b_2 \times x_2 = 0 \tag{4.63}$$

$$b_0 + b_1 \times x_1 + b_2 \times x_2 > 0 \tag{4.64}$$

$$b_0 + b_1 \times x_1 + b_2 \times x_2 < 0 \tag{4.65}$$

Separating The Data

Data separation is done based on the data points which shared the same property. As observed in Figure 4.30, the green- and orange data points receive two properties, respectively. The generalized formula for separating the points can be calculated with Equation 4.66. But in most cases, datasets do not look like this. Most of them will have data points on the wrong side of the hyperplane, and this is where support vectors and margin come in for aid.

$$y \times (b_0 + b_1 \times x_1 + b_2 \times x_2) > 0 \tag{4.66}$$

4.3.2 Support-Vectors

Support vectors are the points that lie closest to the decision boundary (the hyperplane). Those data points are the most challenging for the algorithm to classify. We can see the data points marked with a circle in Figure 4.32. Those data points directly affect the hyperplane, and any modifications will change the hyperplane. Any other data points besides support vectors will not impact the hyperplane. The point of the support vectors is to maximize the margin between the hyperplane and support vectors on both sides.

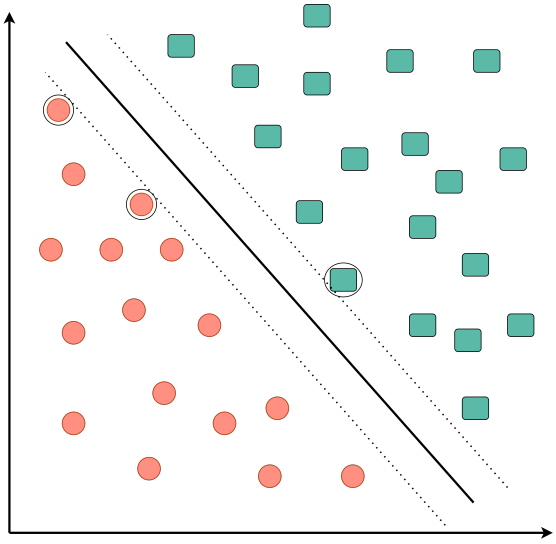


Figure 4.32: Support Vectors

4.3.3 Margin

Margin is the distance between the hyperplane and the support vectors on each side. The margin is the ”line” that makes sure that new data points are on the right side of the hyperplane, where the SVM algorithm tries to maximize this margin. The dotted line in Figure 4.32 shows the margin on both sides.

Cost Function And Gradient Updates

We need to maximize the margin between the hyperplane and the data points. To achieve this, we use a loss function, for example the Hinge Loss function, which is mathematically represented in Equation 4.67.

$$l(y) = max(0, 1 - y^i \times (x^i - b)) \tag{4.67}$$

Here y^i and x^i refer to the instance in the training set, b is the bias, and 1 is the loss of any given instance. So, breaking it down, we get Equation 4.68.

$$l = \begin{cases} 0 & \text{if } y \times (w \times x) \geq 1 \\ 1 & -y \times (w \times x) \text{ otherwise} \end{cases} \tag{4.68}$$

The Hinge Loss function Cost function is optimized explicitly for SVM. Hinge loss separates the boundary into -1 on the left side and $+1$ on the right side. So when a point is predicted, it is checked against the actual value. So if the actual value is $+1$ and the predicted value is -0.25, then the hinge loss equals 1.25. Figure 4.33 shows a visualization of the hinge loss. The y axis represents the amount of loss, and the x-axis represents the distance from the boundary line or hyperplane. The gradient updates are

only done when a data point is misclassified. Then the loss is included as a regularization parameter, and then it updates the weights. If there is no misclassification, the gradient parameters stay the same.

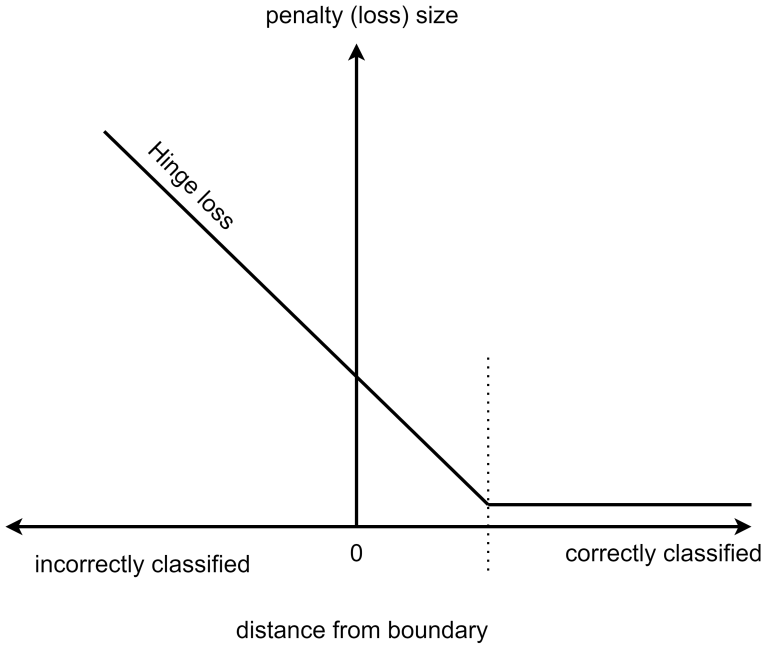


Figure 4.33: Hinge Loss

4.3.4 Linear With Non-Separable Cases

Most realistic datasets cannot be 100% divided into two classes without having some sort of misclassification. So when the hyperplane tries to separate the classes, it needs either let the misclassification happen to some degree or use the kernel trick. Firstly, we will discuss the results if we let misclassification happen.

Soft Margin

The soft margin allows misclassification to happen, but the error still needs to be minimized. We also have to deal with more constraints when dealing with misclassification errors. Therefore, we also need a Loss function, where one of the more used ones is called the Hinge Loss function, which is mathematically represented in Equation 4.68. The C parameter is the one that controls the trade-off between how much loss it should have and maximizing the margin.

Degree Of Tolerance (C)

The C is a parameter in SVM which determines the tolerance of the decision boundary; in other words, the C parameter is the degree of tolerance. This is important for both

linear and non-linear data. The higher we set the C value, the more the penalty the algorithm gets when it misclassifies. A smaller margin will have fewer support vectors to depend on.

In Figure 4.34, we can see that the hyperplane did not separate all the training points correctly. This is because the C value is low, which can lead to misclassification. However, a higher C value does not necessarily result in a higher accuracy. This statement is discussed followingly.

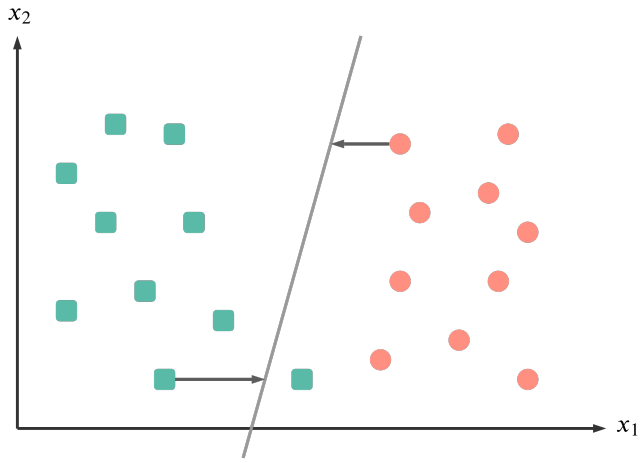


Figure 4.34: Low C-value With No Additional Predictors

In Figure 4.35 the C value is higher than in the last Figure 4.34, and we can observe that the hyperplane successfully separated all the training points.

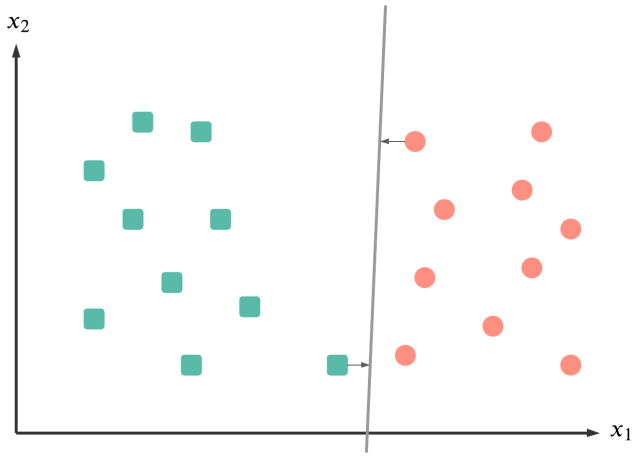


Figure 4.35: High C-value With No Additional Predictors

In Figure 4.36, we can observe a equally low C value as in Figure 4.34, but with more predictors added, where some of them are misclassified due to the low C value.

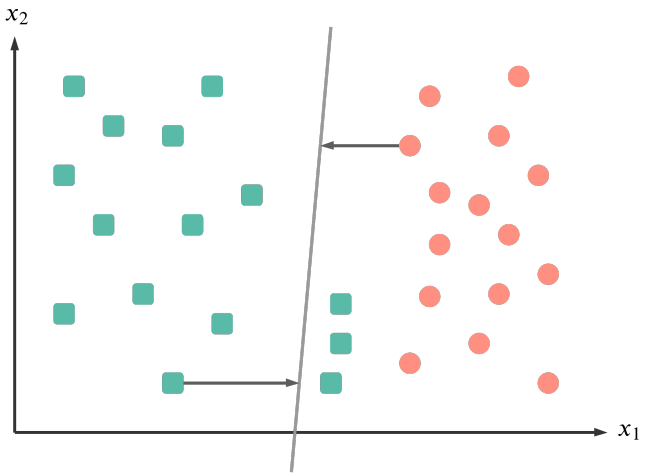


Figure 4.36: Low C-value With Additional Predictors

In Figure 4.37, it is the same C value as in 4.35, but here the hyperplane has successfully separated all the training points and also separated the new predictor points.

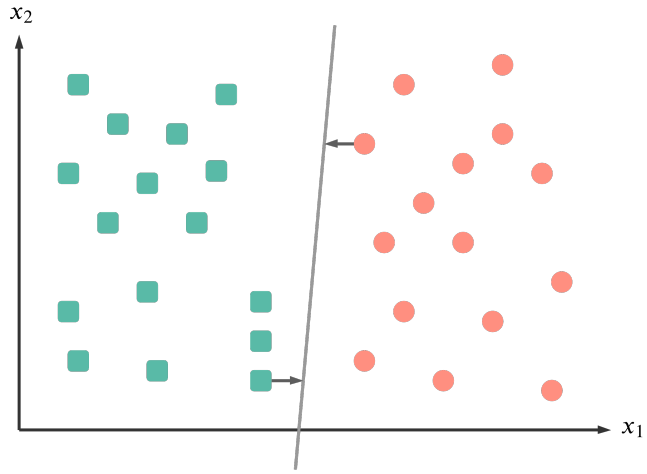


Figure 4.37: High C-value With Additional Predictors

In Figure 4.38, we see that low C values have done an excellent job of separating the new predictors. The only misclassified element is the one data point from the training set. This shows that the C value does not necessarily need to be as high as possible to get the best results.

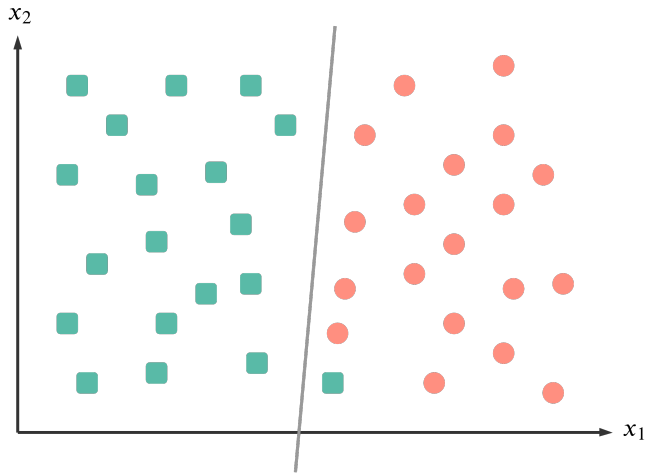


Figure 4.38: Low C-value With Even More Additional Predictors

In Figure 4.39, we have the same predictors and train points as in Figure 4.38, but with a high C value, and here, some data is misclassifiedd. So this shows that determining a C value is a trial and error process, to find the optimal value that separates the training data and testing data.

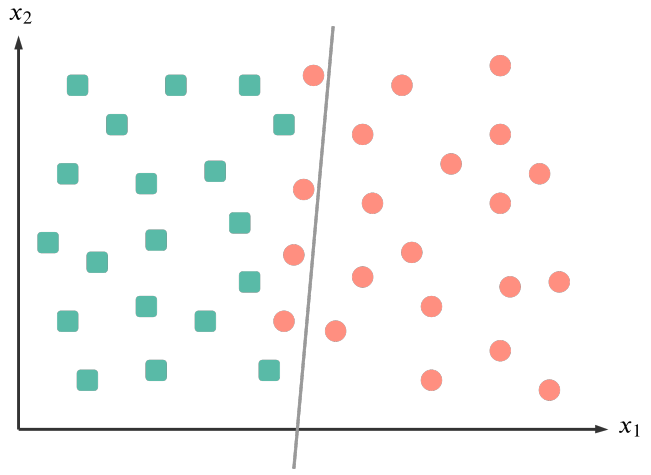


Figure 4.39: High C-value With Even More Additional Predictors

4.3.5 Nonlinear Data

In Figure 4.30, a line can separate the data into two classes. However, when observing slightly more complex dataset, as shown in Figure 4.40, it is clear that no line can separate this dataset without going through a process to change the dimensions of the data. So to work around this problem, the SVM maps the dataset into a higher dimension.

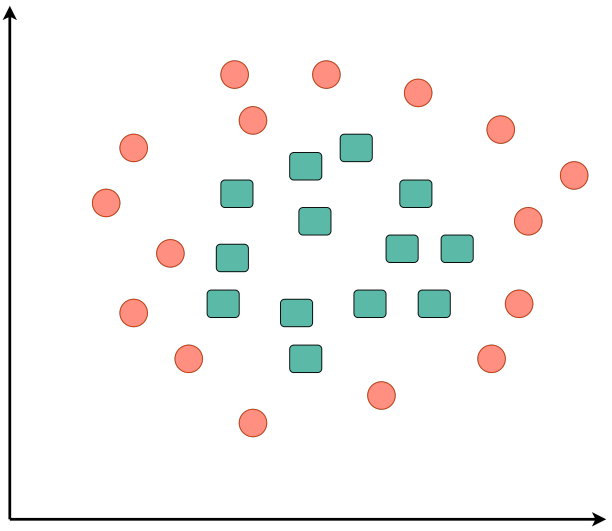


Figure 4.40: 2D Non Separable

Mapping Into a Higher Dimension

To map the example in Figure 4.40, to a higher dimension, it adds a dimension z which will give example shown in Figure 4.41, a way to separate the data with a line. This process is called the Kernel Transformation (the Kernel Trick).

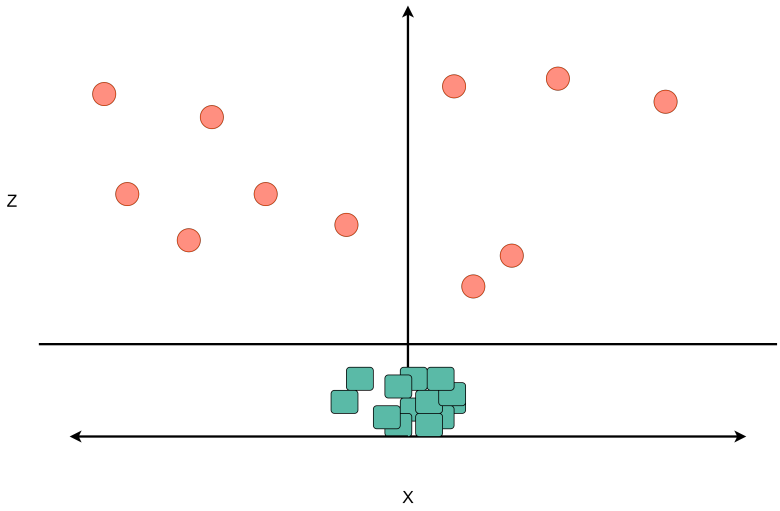
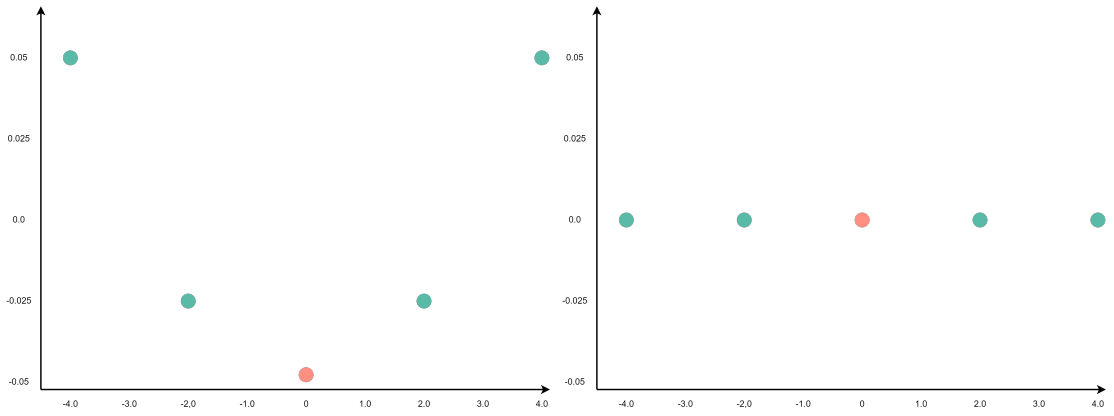


Figure 4.41: Separated Data Points When Added z Dimension

4.3.6 The Kernel Trick

SVM uses kernels which are mathematical formulas that take low dimensional input space and transform them into high dimensional space. There are different kernels like



((a)) Polynomial Separable ((b)) Raw Dataset

Figure 4.42: Polynomial Effect

Linear, Polynomial, Radial Basis Function (RBF), and Sigmoid. Many more kernels are available for SVM, but the mentioned kernels are the most frequently used, hence the only ones included in the Scikit-learn library. RBF is usually the most preferred kernel[95]. The scalar product between two points in a very appropriate feature space is returned by the kernel functions. The different kernels are just different ways of making the hyperplane decision boundary between the classes. The linear and polynomial kernels are less computational-heavy than RBF, but they also usually provide worse accuracy than the RBF or other Gaussian kernels. For the kernel formulas, x and x_i are the data we are trying to classify.

Linear Kernel

The linear kernel is the simplest of the kernels and can only do linear separation of data. It is mainly used when there is a high amount of features in text classification problem. The linear kernel is quicker, and relatively simple compared to the other kernels. When tuning the linear kernel, only the C parameter needs to be taken into consideration. The linear kernel is represented in Equation 4.69.

$$K(x, y) = (x^t \times y + c) \tag{4.69}$$

Polynomial Kernel

The polynomial kernel is a sort of transformer which generates new features. This is done by using the polynomial combination of all the features we already have. The equation for the polynomial kernel is represented in Equation 4.70.

$$K(x, y) = (\alpha \times x^t \times y + c)^d \tag{4.70}$$

For example, a line with x and y, where $x = [-2, -1, 0, 1, 2]$ and $y = [1, 1, 0, 1, 1]$, as shown in Figure 4.42(b). We have five points that lay on a line, which is impossible to separate. By using the transformation x^2, we get new points. The new point will then be $x = [4, 1, 0, 1, 4]$, now we can see that it is possible to separate the green dots from the red dot as illustrated in Figure 4.42.

Radial Basis Function (RBF) Kernel

The radial basis function kernel is a process that generates new features by calculating the distance between all data points to a specific data point's center. The Gaussian radial function is the most common of the RBF kernels[95]. The radial basis function is represented in Equation 4.71.

$$K(x, y) = exp(-\gamma||x - y||^2) \tag{4.71}$$

The gamma parameter is responsible for controlling the influence on the new features of the $K(x, y)$ decision boundary. The higher the gamma parameter is, the more it will influence the features on $K(x, y)$, and the boundary will be more versatile. Observing the exact Figure as on the polynomial kernel with the straight line (Figure: 4.42(b)), it's still impossible to separate them with a line. But with the RBF kernel, we can get a new graph as shown in Figure 4.43, where we can separate the data points.

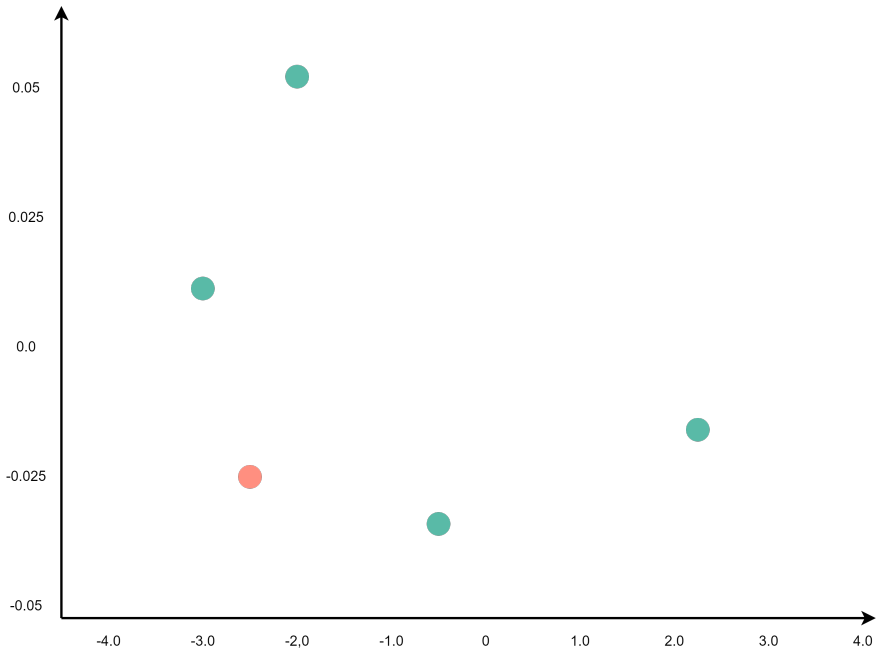


Figure 4.43: Radial Basis

Sigmoid Kernel

The Sigmoid kernel (hyperbolic tangent kernel) is from the neural network algorithm, which is used as an activation function. The Sigmoid kernel works like a two-layered perceptron neural network. The Sigmoid kernel is represented mathematically in Equation 4.72. There are two parameters to tune in the Sigmoid kernel, the C value and the alpha value. The Sigmoid kernel have been quite popular in SVM due to it coming from ANN [96].

$$K(x, y) = tanh(\alpha \times x^T \times y + c) \tag{4.72}$$

4.4 Decision tree (DT)

Decision trees (DT) is a supervised ML algorithm used for regression and classification problems. DT is one of the most popular ML algorithms out there[97]. The objective is to make a model that can predict a value based on a target variable. To make the model, it first needs to make rules based on the training features. The model is easy to understand and interpret, and multiple tools can be used to visualize it. It is often utilized for multiple tasks, such as ML and data mining.

4.4.1 Terminology

Table 4.5: Decision Tree Terminology

Terminology	**Description**
Root node	The root node is a representation of the whole dataset/population. The root node will then be divided into multiple homogeneous sets based on the dataset's size.
Leaf node	A leaf node also can be called a terminal node. This is where nodes stop splitting when there is no more data to split.
Decision node	A decision node is where a sub-node is split into more sub-nodes.
Splitting	Splitting is when the current node is divided into two or more sub-nodes.
Branch/sub-tree	A branch or sub-tree is a section in the entirety of the tree.
Pruning	When pruning a DT, it removes a sub-node of a decision node. It is like the opposite process of splitting.
Parent	A parent node is the node of the sub-nodes, and the subnodes are the child node of the parent node.

Figure 4.44 shows an overview of the terms mentioned in Table 4.5.

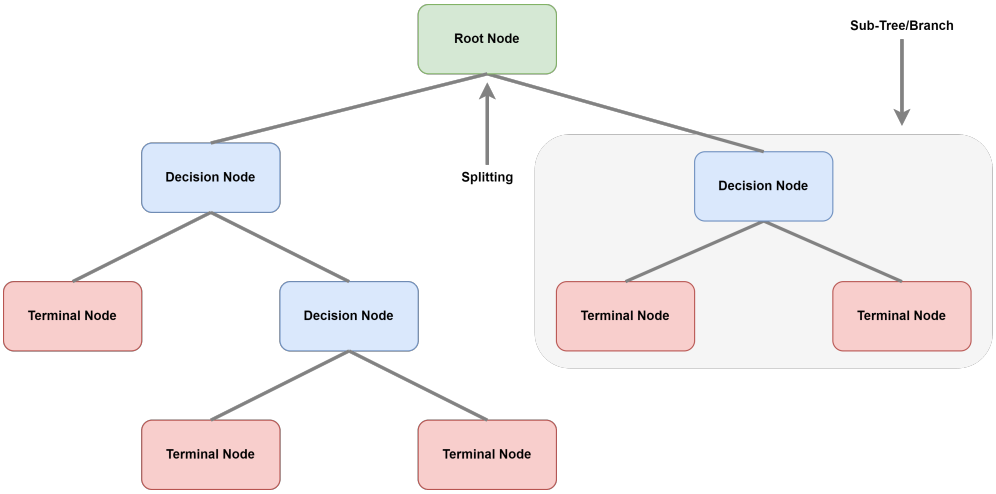


Figure 4.44: Decision Tree Terminology

4.4.2 Assumptions When Creating a Decision Tree

When creating a DT, there are some assumptions to consider. Before anything is done, the whole training set is considered the tree's root. The DT algorithm prefers categorical values over continuous values. If continuous values are going to be used, they need to be discretized[9] before building the model. Every attribute record is distributed recursively based on the values each attribute holds. The DT algorithm places the root node or an internal node by applying a statistical method to it, explained in Section 4.4.3. When representing a DT, it uses the Sum of products (SOP). SOP is also known as the Disjunctive Normal Form. The product part is every branch from the leaf node to the root with the same class in conjunction with values. The sum is the different branches that end in the same class form. The main goal of the DT algorithm is to find out which attribute provides the most information and make that the root, and each level below the root. There are several ways to find out which attribute gives the most information—some of the most common ways to make attribute selection are described in Section 4.4.3.

4.4.3 Attribute Selection Measures

Attributes are considered as the root, in the beginning. An attribute selection method needs to be applied when deciding which attribute is the root and which goes on each level. If this is done by selecting random attributes, the performance will likely be inadequate. There are multiple attribute selectors used for different tasks explained below. They calculate the values of each attribute and sort them from highest to lowest. In the case of Information Gain (IG), the highest value will be the root. Gini index attributes will be continuous, and IG the attributes will be categorical.

[9]The process of mapping values in to discrete values. Example 0.0-0.5 -> 1, 0.5-1.0 -> 2 and so on

Information Gain (IG)

Information Gain (IG) is a statistical property used to make a DT. The higher the IG, the better the separation is. IG should be high, and entropy should be as small as possible. IG is a way to separate attributes in the training examples based on the target classification. In this Figure 4.45, we can observe that the figure to the right got even splits in both nodes, giving a low IG. In the figure to the left, the IG is high because the content in the nodes is uneven.

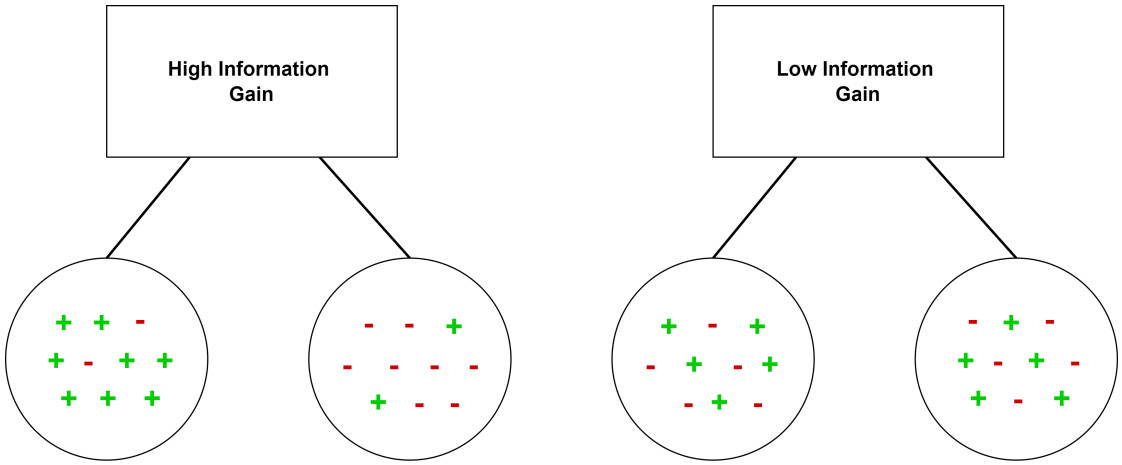


Figure 4.45: Information Gain

IG is represented in Equation 4.73. When IG is calculating, it takes the entropy from before the split and the average entropy from after the split. $Entropy(T)$ is the dataset before the split, and K is the number of subsets made by the split. Furthermore, (T, X) is the subset after the split.

$$InformationGain = Entropy(T) - \sum_{j=i}^{K} Entropy(T, X) \tag{4.73}$$

Gini Index

Gini index is used by the Classification and Regression Tree (CART) algorithm, which is Cost function that is used to check if a split in a dataset is good or not. The Gini index can be calculated with Equation 4.74.

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2 \tag{4.74}$$

It takes the squared probability for every class. When the squared probability is calculated, subtract it from 1 to get a value. The higher the value gets, the higher the heterogeneity. Gini index only works with categorical target variables and only performs binary splits. The target variable is either failure or success.

Entropy

Entropy is information about how random the current data is. If the entropy is high, it is hard to guess the outcome. Figure 4.46 shows that when the x-axis is in the middle, the entropy is at the highest, giving the outcome a 50/50 for either outcome, while far to the right and far to the left, the outcome will be corresponding to the side the value resides. The Entropy can be calculated with Equation 4.75.

$$E(S) = \sum_{i=1}^{c} -p_i \times \log_2 \times p_i \tag{4.75}$$

Here P_i is the most frequent element in the class. Furthermore, it can be either $-$ or $+$ depending on the split in the class. Entropy is also used to calculate IG.

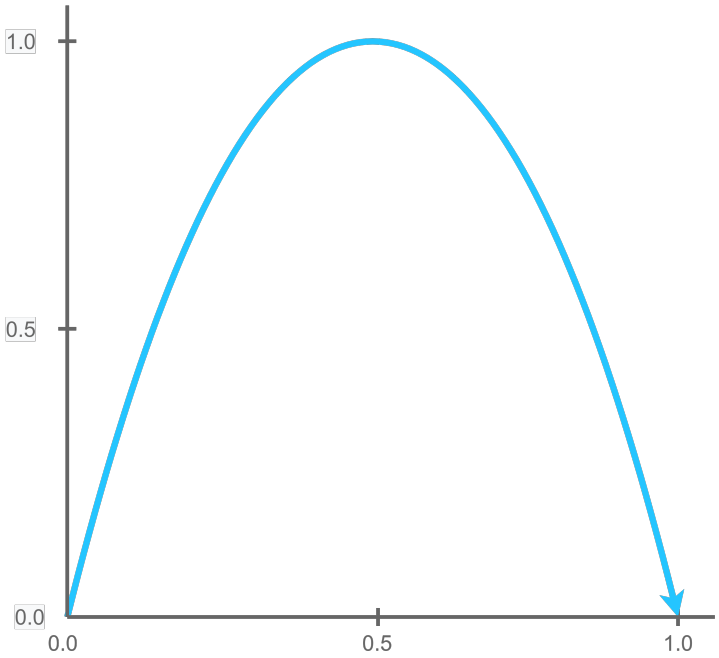


Figure 4.46: Entropy Perfect Randomness

Gain Ratio

The C4.5 algorithm is an improved version of the ID3. It uses Gain Ratio instead of IG, which is a modified IG. IG tends to bias towards attributes with many values for the root node. The Gain Ratio reduces this bias. It deals with this problem by taking the number of branches into account before the splits. Gain ratio, calculated with Equation 4.76, uses IG as part of the equation, and it's divided by Intrinsic Information, shown in Equation 4.77.

$$GainRatio = \frac{InformationGain}{IntrinsicInformation} \tag{4.76}$$

$$IntrinsicInformation = -\left(\sum \frac{T,X}{T} \times \log_2 \times \frac{T,X}{T}\right) \tag{4.77}$$

The Intrinsic Information is the entropy of the sub dataset's distribution.

Reduction in Variance

Reduction in variance is an algorithm used for regression problems where the target variables are continuous. It uses the standard formula for calculating the variance and uses this value to choose the best split. The one with the lowest variance is the one that is chosen as the common value to split the population. Variance is represented in Equation 4.78.

$$Variance = \frac{\sum(X - \overline{X})^2)}{n} \tag{4.78}$$

The Reduction in Variance first calculates the variance for each node. Then it calculates the variance for each of the splits. This is done based on the weight of the average node variance. Then it selects the split with the lowest variance. This process is recursive until all the nodes are homogeneous.

Chi-Square

Chi-Square is used for the Chi-Squared Automatic Interaction Detector. It is one of the oldest DT classification methods. The Chi-Square works by taking the difference between the child nodes and the parent nodes and measuring it by the sum of the square of the standardized differences between the observed outcome and the expected outcome. If Chi-Square receives a high value, it gets a high statistical significance of the difference between the parent and child nodes. The Chi-Square is calculated by Equation 4.79, where O is the observed score, and E is the expected score.

$$\chi^2 = \sum \frac{(O - E)^2}{E} \tag{4.79}$$

Chi-Square first needs to calculate the individual node. To do this, it takes the deviation for both success and failure. Then it takes the sum of the chi-squares from both failure and success splits and calculates the Chi-Square for an individual node.

4.4.4 Different Algorithms Used In Decision Tree

Since DT can do both regression and classification, it has multiple algorithms based on the target variable it will predict. Those algorithms decide how the nodes will be split and the number of splits. A node's purity[10] increases based on the target variable. The splits will happen on all available variables, and then it finds the split that gives the most homogenous sub-nodes. Some of the most common algorithms used in DT are: ID3, C4.5, CART, CHAID, and MARS.

[10]If a node is pure all of the data points belong to a single class. And if a node is impure then there is a even distribution in the class.

The Iterative Dichotomiser 3 (ID3)

The Iterative Dichotomiser 3 (ID3) algorithm is used to make a DT and is the precursor to the C4.5 algorithm. The ID3 algorithm was invented by Ross Quinlan in 1986[98]. ID3 algorithm starts with the raw dataset S and sets that as root node. The algorithm then iterates through the entire dataset for every unused attribute of dataset S. Then it calculates the Information Gain($IG(S)$) or entropy($H(S)$) of the attribute. It selects the one with the most significant IG or smallest entropy from the calculated values. The training set S is then separated by the attribute selected by the calculations, then it produces a subset of the data. The node is then split into child nodes based on the population of the node. The algorithm keeps recursing for each subset until it's done, and it is only considering unused attributes so far in the algorithm. The recursion stops when:

1. When all elements remaining in a subset belong to the same class, in this case, the node is turned into a terminal node and gets labeled with the class name.

2. When all of the attributes have been selected and all of the remaining examples do not belong to the same class. If this happens, the node is made into a terminal node, and the example of the most common case will be the label.

3. When there is no example in the parent node's set that's found and matches a value of the selected attribute, if this is the case, then a terminal node is created. The label of this node will be from the parent node's most common example.

C4.5

The C4.5 algorithm is an extension of the ID3 algorithm and is developed by the same author Ross Quinlan[99]. This algorithm is mainly used for classification and is often called a statistical classifier. C4.5 algorithm builds the tree the same way as ID3 with IG and entropy. The dataset S consists of samples that are already classified. Each sample of S_n has a dimension vector where each of those vectors has an attribute value of the sample and a class it falls into. The algorithm chooses the attribute that most effectively separates the dataset into subsets, which is done for each node in the tree. The splits are based on the IG, and the IG value has to be normalized for the algorithm to work. The highest value here will be responsible for making the decision. This process recurses until it is done. It has a few base rules it follows for when something does not fit elsewhere.

1. If all the values in a sample belong to the same class, it creates a terminal node for that DT.

2. If the information gained from a feature is irrelevant, the algorithm makes a decision node higher up in the DT and uses the expected value from the class.

3. If a class that previously has been unseen occurs, it will do the same as when the IG is irrelevant.

Classification And Regression Trees (CART)

The CART algorithm is binary tree, which was developed in 1986 by Leo Breiman[100]. The CART algorithm is used for classification, and the criterion that it uses is the Gini impurity index, as shown in Equation 4.74. The CART algorithm has three main steps:

1. First, the algorithm has to find the best split. The amount of splits possible is less than the number of unique features. It has to find the one that gives the maximum split for the splitting criterion.
2. Then it has to find the split for the current node. Use the best split from step 1.
3. Then lastly, split the node bases on the best split found in step 2. Repeat the steps until the stopping criteria are reached.

Chi-Square Automatic Interaction Detection (CHAID)

CHAID is the oldest DT algorithm and was developed by Gordon V in 1980[11][101]. This algorithm is used for classification problems, and it expects that the dataset it receives has a categorical target variable. It uses a Chi-Square, as shown in Equation 4.79, to test and find which of the feature is the most dominant in the entire dataset. For regression problems, it uses F-test[12]. The algorithm can take different data as input, such as ordinal, continuous, nominal, and analysis data.

The CHAID works algorithm by:

1. Firstly it has to find a variable considered as the root node. This can be either a dependent variable or the target variable.
2. It builds a tree to predict the outcome of a dependent variable based on how the given variables merge. When the algorithm gets the data, it splits the continuous predictors into categories where the number of observations is about the same.
3. Then all the predictors are cross-tabulated to find the best results, and it is impossible to split the predictors further.
4. CHAID splits the target variable into two or more parent nodes. The parent nodes are then split into child nodes using a statistical approach.

Multivariate Adaptive Regression Spline (MARS)

Jerome H. Friedman developed the MARS algorithm in 1991[102], [103]. The MARS algorithm is used for regression, to predict continuous target variables. MARS algorithm is based on Linear regression, which assumes a relationship between output and inputs in most cases. However, since most real-life examples cannot be linearly separated, linear

[11] `https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/chaid/`

[12] A F-test is used to evaluate a hypothesis. It checks the quality of two variances from different classes

regression rarely works on its own. This is where the MARS algorithm comes in. MARS is a sort of ensemble method but for Linear functions along with a Hinge function, as shown in Equation 4.80.

$$h(x-c) = max(0, x-c) = \begin{cases} x-c & \text{if } x > 0 \\ 0 & \text{if } x < c \end{cases} \tag{4.80}$$

Here c is the constant. c is known as a knot. An example of combined Linear Hinge function is shown in Figure 4.47. The green dots in this figure are the observations done, and the blue line is the algorithm's prediction. Here, it is clear that no single line can separate this data.

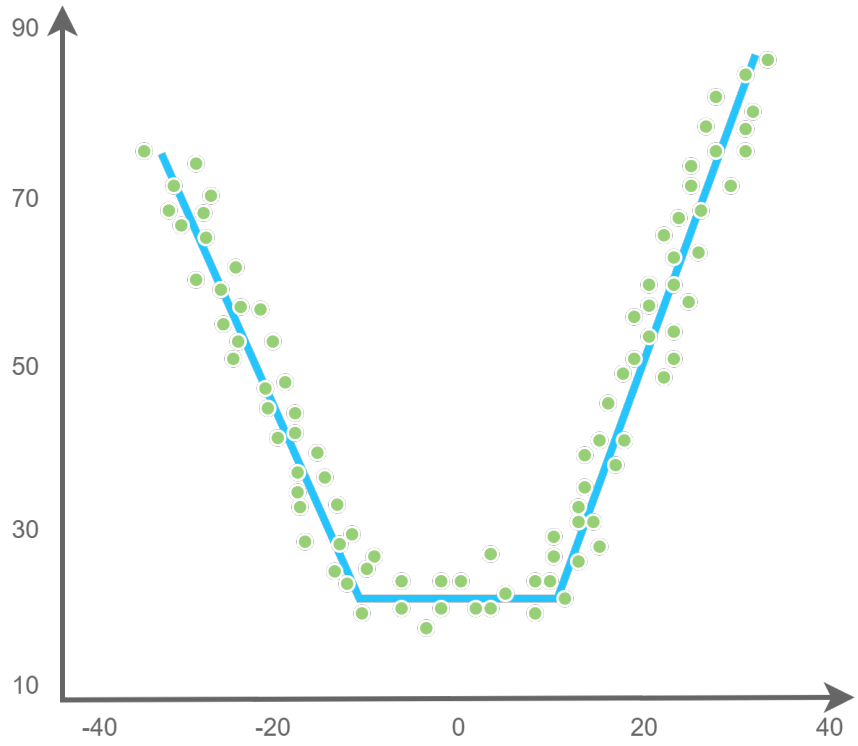


Figure 4.47: Hinge Function

So the MARS algorithm combines multiple linear functions by using hinges. The MARS algorithm operates in two stages: backward and forward stages. The forward stage generates the candidate's functions, which are produced in pairs and will only be added to the model if its error is lower than before. The number of features can be controlled with a hyperparameter. The backward stage is a pruning stage. The functions that do not give anything in the form of performance will be deleted. It finds out which function does not give any to the model by using an approximation of the Cross-Validation score. The goal of this score is to penalize how complex the model is. By performing the two

steps, the algorithm will find multiple linear functions which can be put together to split the data as shown in the Figure 4.47.

4.4.5 Training The Decision Tree

Algorithm 2 shows the process of training the DT algorithm. Firstly it checks if the stop criteria have been accomplished. If the stop criteria have not been reached, all the attribute splits will be calculated based on their value, and the nodes will be labeled according to the optimal split value. Then it proceeds to split the nodes into multiple nodes, one node for each of the unique values in the specific attribute. Then it recurses on the same process over and over on all of the subsets until it reaches the stop criteria. This will contain all the data with the matching values from the chosen attribute. If it has reached the stop criteria, the current node will be labeled with the most common value in the training set out of all the classes.

Algorithm 2: Decision Tree[104]

Data: Training Set S, Input Feature Set A, Target Feature y, Split Criterion $Split_C$, Stop Criterion $Stop_C$,

```
GrowTree(S, A, C, Split_C, Stop_C)
if Stop_C(S) = False then
    for all a_i ∈ A do
        find a_i with best optimized Split_C(S)
    end
    label current Node with a
    for all values v_i ∈ a do
        label outoing edge with v_i
        S_sub = S where a = v_i
        create subNode = GrowTree(S, A, C, Split_C, Stop_C)
    end
end
else
    CurrentNode = Leaf
    label currentNode with c_i where c_i is most common value of C ∈ S
end
```

4.4.6 Pruning Decision Trees

DT can easily overfit if there are no limits to the tree. The tree will have 100% accuracy by making a single leaf for every observation in the dataset. If the tree has one leaf for every observation in the dataset, the predicting of testing and validation sets will be affected. A way to stop overfitting DT is to prune them.

Pre-Pruning

Pre-pruning brings the DT to an early stop by giving it constraints, which it then evaluates. Those constraints can either be from the IG or gini impurity measures. The conditions are measured at each node in the tree. If one of the conditions is met, the subtree will be pruned. By pruning, a tree replaces a decision node with a terminal node. The advantage of pre-pruning is that it stops the model from overfitting, which is a big issue for DT. It also increases the model's efficiency and makes it faster to process.

Post-Pruning

Post-pruning is pruning the tree after it has been built. The tree builds the defined model as usual, and then it gets pruned from the bottom and up. Post-pruning is based on the same measures as pre-pruning, either Gini impurity or IG. This process starts at the decision node at the bottom of the tree and decides whether or not to keep it. If the algorithm decides not to keep it, the decision node will be changed to a terminal node. This process can take away a subtree and replace it with a single terminal node.

Cost-Complexity Pruning

A pruning technique that goes under the post pruning process is cost-complexity pruning. This technique calculates a score of the tree with the use of Equation 4.81, where y_i is the value of the predicted variable, $f(x_i)$ is the predicted value of y_i and n is the upper limit. The TreeScore, shown in Equation 4.82, is calculated by taking the Residual Sum of Squares (RSS)[13] and alpha score[14] and the number of leaves in a subtree.

$$RSS = \sum_{i=1}^{n} (y_i - f(x_i))^2 \tag{4.81}$$

$$TreeScore = RSS + \alpha \tag{4.82}$$

The TreeScore is calculated for all subtrees in the entire tree. Then it picks the subtree with the lowest score. It is essential to find the correct alpha score. The algorithm executes Cross-Validation to find an alpha score. Then the process above is repeated with a different alpha score from the cross-validation until it has a sequence of trees. Each of the tree's TreeScores is calculated, and the one that gives the lowest score averagely will have its alpha score recommended.

4.4.7 Hyperparameters

Table 4.6 represents the different hyperparameters for DT.

[13] RSS is a equation used to measure the variance in the data

[14] The alpha score keeps track of the weakest nodes, and those are the first to be pruned

Table 4.6: Decision Tree Hyperparameters

Hyperparameter	Description
criterion	The criterion parameter chooses which function the algorithm uses to measure the split. Entropy and log_loss are used for the IG approach, while "gini" is used for the Gini impurity loss.
splitter	The splitter chooses what kind of method to use to split each node. The choices are between random and best splits.
max_depth	The max_depth parameter decides the depth of the tree. If this parameter is undefined, the tree will grow until all the leaves contain less than the min sample split's samples or until they are pure.
min_samples_split	This parameter can be an int or a float number. If it is an int, it is considered the minimum amount of samples required to split a node. If it is a float number, it will be a fraction of the minimum number of samples for each split.
max_leaf_nodes	This defines the number of leaf nodes the tree can have. The best nodes are the ones with the best impurity ranking.
max_features	is a parameter to decide the number of features it considers when looking for the best split. It can be integer, float, log2, or sqrt.
ccp_alpha	This parameter is a part of a pruning process often used as a pre pruning process. The cost-complexity pruning process finds a float number bigger than a subtree's complexity cost.

4.4.8 Knowing The Advantages And Disadvantages

There are some advantages and disadvantages with DT. Knowing the limitation of the algorithm makes it easier to know when to use it and when not to use it. Alternatively, what to do with the DT in some cases.

Advantages

Decision tree is a robust algorithm with many advantages, such as:

1. It works with categorical and numerical data and variables.
2. It does not need as much data cleaning as some other ML algorithms to work well.
3. It is a simple algorithm to understand and got lots of tools that help visualize the tree.
4. It can solve problems with many different outputs.

Disadvantages

Decision tree got some weaknesses that are good to keep in mind before choosing an algorithm for a dataset.

1. DT is highly affected by noisy data.
2. It does not deal with big datasets very well.
3. If the outcome of the dataset is uncertain or linked with other outcomes, the tree can become very complex.
4. DT also only deal with binary outcomes, so that can reduce the amount of complexity.
5. It can easily overfit, but this can be reduced by doing some pruning or using RF instead.

4.5 Random Forest (RF)

Random Forest (RF) is a robust algorithm for regression, classification, and other tasks which involves constructing a significant amount of DT when training. RF is an ensemble learning method developed by Leo Breiman and Adele Cutler in 2006 [105], but prior to this, Tin Kam Ho developed the first Random Decision Forest in 1995 [106]. In 2009 the trademark is now owned by Minitab, inc [107].

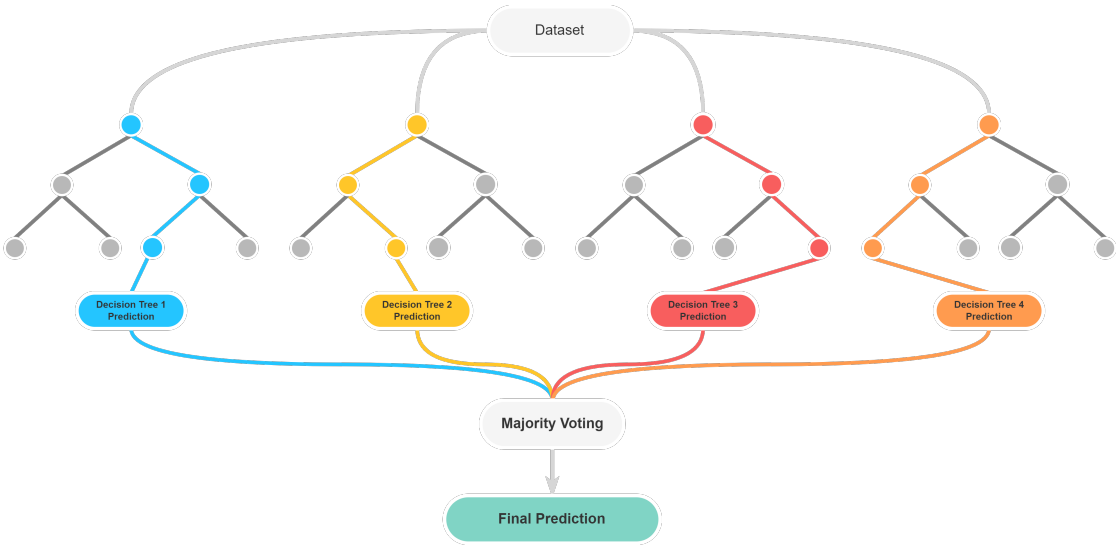


Figure 4.48: Random Forest Classifier

4.5.1 Preliminaries

RF is an ensemble method that uses DTs to make a forest as shown in Figure 4.48. So to understand the RF, one got to have complete knowledge of the DT algorithm (Refer to Section 4.4 for more information). Using RF solves one of the significant issues DTs have, which is overfitting. A DT can grow very deep, which causes the overfitting of the training sets. RF uses the same dataset for multiple DTs. Each DT is trained on different parts of the training set, and then it takes the average score from the trees to reduce the variance. This technique usually increases the model's performance, but it can increase bias and be harder to interpret.

4.5.2 Bagging

Bagging is also known as Bootstrap Aggregation. Bootstrap Aggregation is the technique that RF applies for its majority voting. Majority voting works by taking random sample from the dataset and generating a model from this sample. This method is called Row Sampling. When every model is trained independently, the results are saved. Then, when all the models are done with their individual performance, a majority vote combines all

the results into one. This sequence is called Aggregation. We can describe Bootstrap Aggregation by an example: each samples shown in Figure 4.49 is selecting data from the original dataset. When selecting from the actual dataset, it is with replacement, so there is a chance for each of the samples to contain some of the same values. Each of the samples is now trained individually, where the results is sent to the ensemble classifier, where the majority vote decides the final outcome.

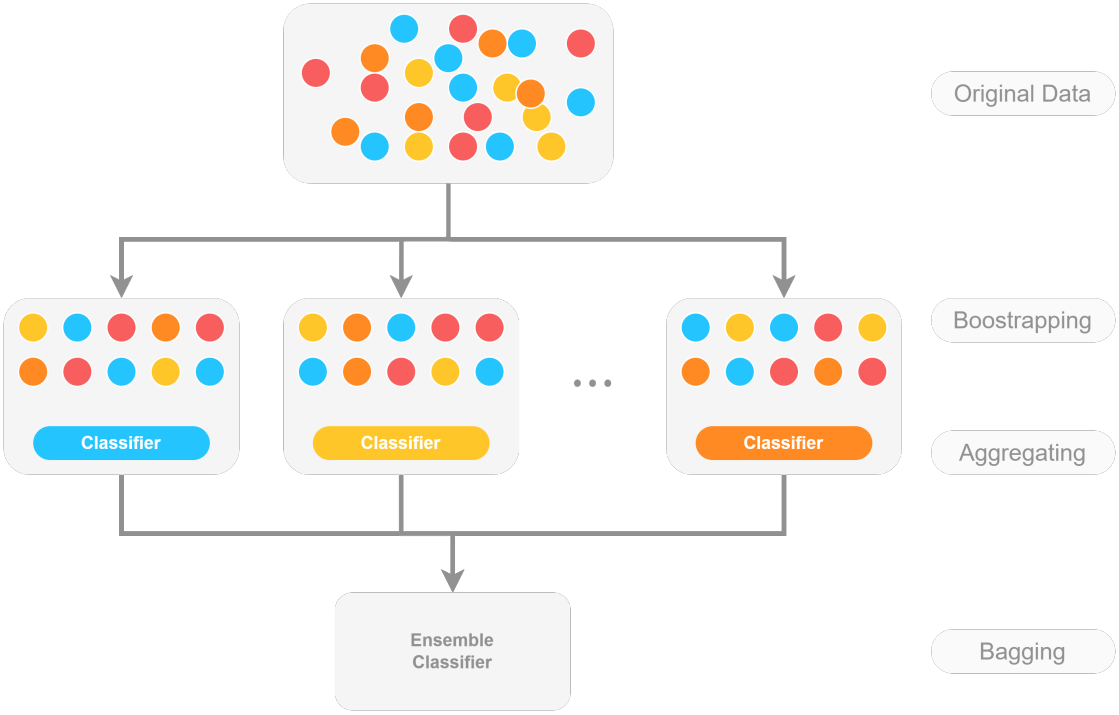


Figure 4.49: Bagging Example

4.5.3 Boosting

Boosting is when an algorithm train weak learners to become strong learners, which is a sequential process. Each model tries to fix the error of the prior model to reduce Bias. Boosting is easily prone to overfitting, which is one of the main disadvantages, but this can be avoided with pruning. The frequently used Boosting techniques are:

- GBM
- XGBM
- CatBoost
- AdaBoost

Summarizing Bagging and Boosting:

1. Bagging is when the training data are split into a subset with replacements, and the final results are based on majority voting, such as RF.

2. Boosting is when the weak learners are combined to make one strong learner. It does this by making the models in a sequential pattern so the last model will have the best performance.

Figure 4.50 illustrates how Bagging and Boosting work.

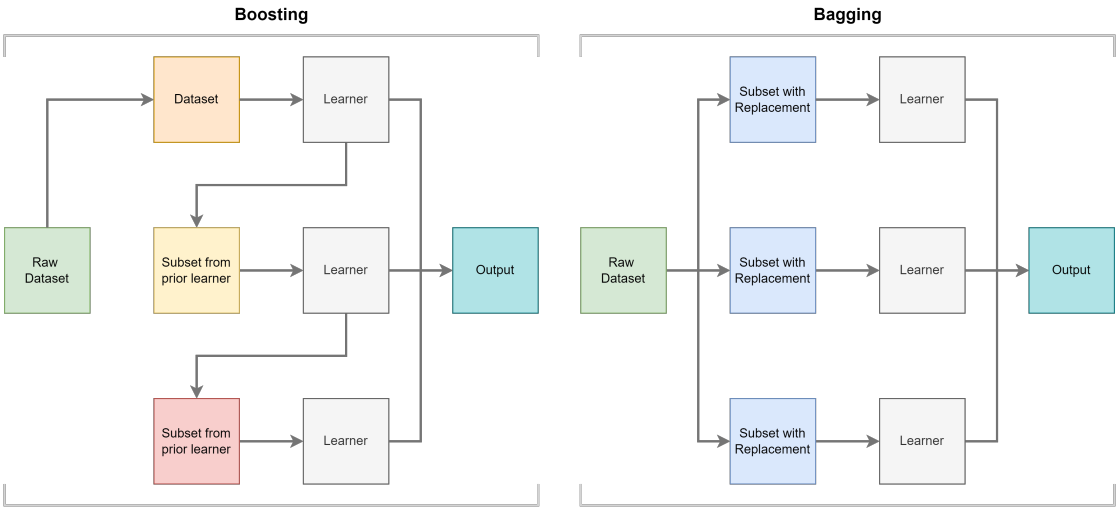


Figure 4.50: Difference Between Bagging And Boosting

4.5.4 Ensemble

The ensemble method's goal is to increase the performance of any statistical learning or a model's fitting technique. Ensemble methods create a combination of linear fitting methods instead of a single fit method. The ensemble method can be trained and used to make predictions, therefore, making it a supervised ML algorithm. It takes a group of weak learners and merges them to make a strong learner, which increases the model's performance. What ensemble methods do is reduce variance and bias errors. Those factors are the ones that cause the difference between the predicted values and the actual values in any given model. Also, noise can affect this outcome, but this is dealt with by reducing the Variance and Bias. All of the errors can be capture for continuous variables with Mean Squared Error (MSE), shown in Equation 4.83.

$$MSE = E\left[\left(Y - f^{(x)}\right)^2\right] \tag{4.83}$$

Y is the actual value, $f^{(x)}$ is the predicted target variable value, and E is the expected mean.

Applying Bagging, as shown in Figure: 4.51, and Boosting techniques to a model can help with reducing the variance and makes the model more robust. It uses a combination

of classifiers, which is good if the classifiers are unstable. An ensemble which consists of a highly unstable single classifier, will result in poor performance.

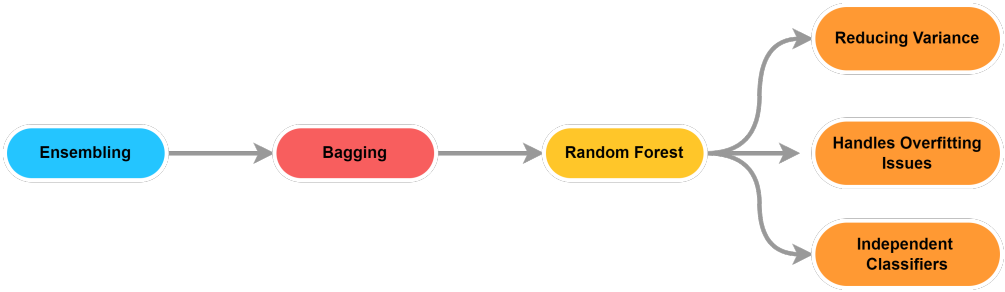


Figure 4.51: Ensemble Algorithm

There are two different methods for ensemble algorithms. First, one is Boosting methods, where and the estimators are built in a sequence to reduce bias on the combined estimator. In comparison, the other method is an average method. This method builds several trees simultaneously and takes the average value from their predictions. The combined tree it creates often performs better and with reduced variance.

4.5.5 Difference Between Decision Trees and Random Forest

Table 4.7 explains the difference between Decision Trees and Random Forests.

Table 4.7: Difference between Decision Trees and Random Forests

	Decision Tree	**Random Forest**
1	Decision trees can easily overfit if the tree is left to grow without any form of pruning.	Random forest uses subsets of the dataset for each tree, and the output from all the trees will be put into a majority vote. This usually solves the overfitting issue in most cases
2	Decision trees are very fast and require less computational power to perform the task	Random forest takes a longer time and uses more computation power to perform the task.
3	When building a decision tree, a set of features will be imported into the tree, and from this, a set of rules will be formulated to make the rules it uses to make predictions	Random forest takes random observations and builds decision trees. The average from those trees will be the prediction it makes. So in the case of random forest, it will not make any rules

4.5.6 Hyperparameters

The hyperparameters for RF are explained in this Table 4.8. Some of the parameters in RF and DT overlap. The overlapping features are explained in the Decision Tree Section 4.4.7)

Table 4.8: Random Forest Hyperparameters

Hyperparameter	Description
n_estimators	Parameter to define the number of trees in RF.
min_samples_leaf	Parameter to define the number of samples required to be in a leaf node at each stage. A split will only be considered at any given depth in the tree if it leaves the min sample leaf number in the right and left node.
n_jobs	States how many jobs are running in parallel. Defining -1 will use all of the cores in the processor to work on this problem.
random_state	Controls the randomness of which features to consider when sampling them for the best split at each node. It also controls how random the bootstrapping is when picking the samples for tree building.
oob_score	is only used when bootstrapping is enabled. This parameter controls if it is going to use out-of-bag samples to estimate the generalization score.

4.6 K-Nearest Neighbor (KNN)

The supervised ML algorithm K-Nearest Neighbor (KNN)[108] algorithm can address both classification and regression tasks[109], [110]. The KNN assumes that similar data points exist in closest proximity meaning that there are near to each other, as shown in Figure: 4.52. Distance metrics such as *Euclidean distance, Hamming distance, Manhattan distance* and *Minkowski distance* are utilized to estimate the distance between points for locating the closest similar points.

KNN can implement Anomaly Detection (also called outlier detection) by categorizing training data as points, and the class with the closest point aggregates it as an input. The algorithm generalizes pleasingly for training datasets with many inputs, where the training set may be expanded at any moment. Significant speed gains may be realized by sorting the data according to each feature and utilizing complicated data structures such as kd-trees[15]. Furthermore, the KNN approach can be enhanced in speed and stability concerning the order of data presentation by substituting the sort operation with the computation of order statistics[80], [111].

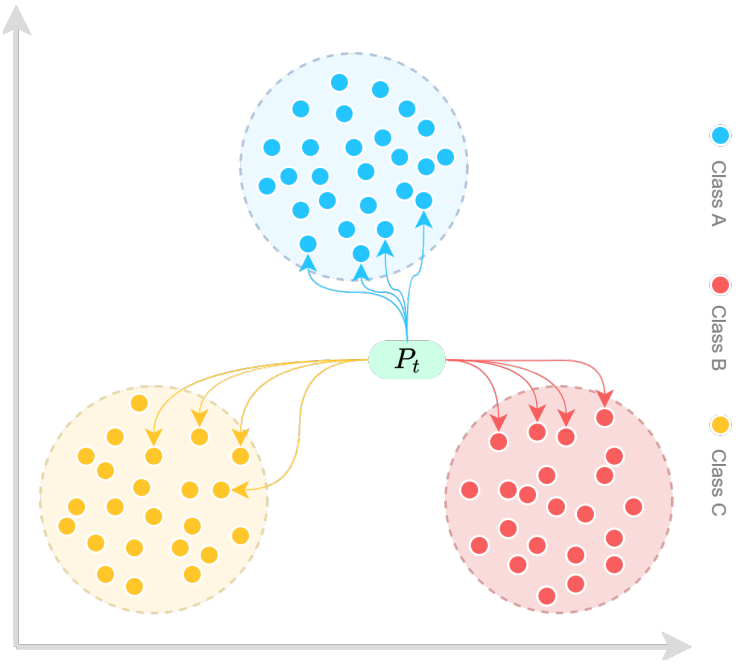


Figure 4.52: K-Nearest Neighbor Classifier

The Euclidean distance (Equation: 4.84) uses the Pythagorean Theorem at heart: $a^2 + b^2 = c2$. In Equation 4.84, x_i and y_i are the coordinators of the i^{th} featured element, respectively, and n is the total number of features in the dataset. The order of the norm is represented by the parameter p in Minkowski Distance (Equation: 4.85). Manhattan

[15] `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KDTree.html`

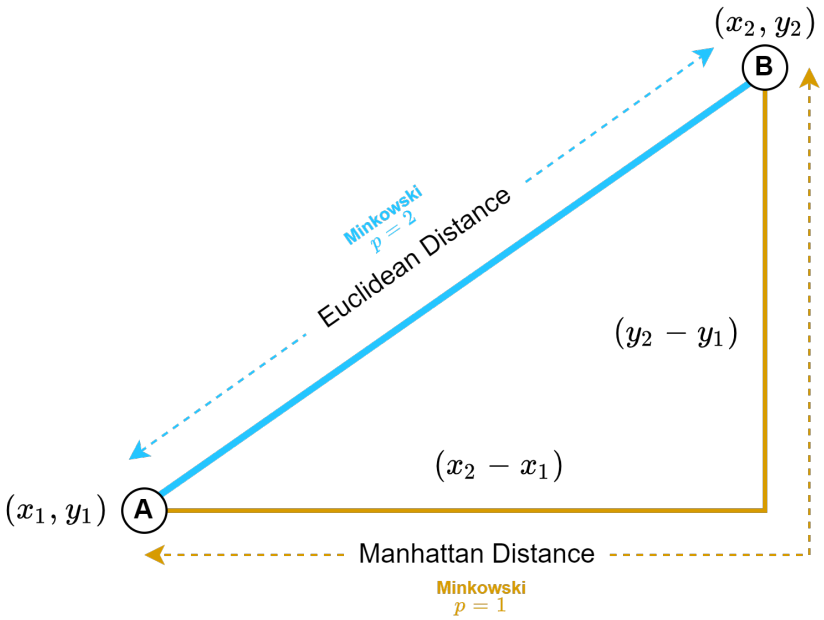


Figure 4.53: Euclidean/Manhattan Distance

distance is represented by defining the order (p) to 1, and when the order (p) equals 2, represents the Euclidean distance. We can utilize the Euclidean Distance, calculated by Equation: 4.84, Minkowski Distance, calculated by Equation 4.85, Manhattan Distance calculated by Equation 4.86, for KNN implementation. Figure 4.53 is visual representation of the Euclidean-, Minkowski- and Manhattan distances.

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{4.84}$$

$$d(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}} \tag{4.85}$$

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i| \tag{4.86}$$

The main advantages of KNN:

1. The algorithm is simple and straightforward to implement
2. There is no need to create a model, tweak parameters, or make any further assumptions
3. The algorithm is highly adaptable. It has classification, regression, and search capabilities.

One of the most significant disadvantages is that the algorithm gets much slower as the number of input/independent -variables increases. Meaning the volume of data has a significant impact on the runtime for the algorithm. Thus, KNN is not suitable for applications where real-time data output is crucial. However, with proficient computer resources, KNN can be very useful for identifying similar objects. The main steps of the KNN is shown in Algorithm 3.

Algorithm 3: K-Nearest Neighbor (KNN)

Data: K-value, dataframe
Result: Class (attack or non-attack)
1 Load preprocessed splitted training and testing datasets.
2 **for** *training_points* **do**
3 | Calculate $d(x, y_i)$ with $i = 1, 2, ..., n$ where d denotes the Euclidean Distance between points (Equation 4.84)
4 | Sort calculated distances in ascending order
5 | Get top K rows from the sorted array
6 | Assign a class to the training data point, based on majority voting
7 **end**
8 **return** *predicted_class*

4.6.1 K-value

We run the KNN algorithm numerous times with different values of K to find the K that decreases the number of errors we encounter while retaining the algorithm's capacity to generate correct predictions when it has given data it has not seen before. There are some aspects to consider when choosing the K value. First, our predictions become less stable as we reduce the value of K to 1 (overfitting and high variance), where the error will always be zero since the algorithm chooses the nearest Class to assign the datapoint. Second, as the value of K increases, our predictions become more stable due to majority voting and averaging, and hence more likely to be accurate, which is up to a certain point. We eventually will start to see an increase in the number of errors and know that we have pushed the value of K too much.

Before implementing the algorithm, we should consider preparing the data through three stages in the preprocessing phase: Data Scaling, Feature Selection, and Missing value imputation (Section: 5.3). We want to scale the dataset with the help of normalization (standardization) so the algorithm can allocate correct predictions in the feature space. If there are too many features, KNN may not operate effectively. As a result, procedures like feature selection and analysis may be used to reduce dimensionality. If one of the feature data for a given sample in the training set is missing, we will be unable to find or compute the distance from that location. As a result, either deletion or imputation is necessary[112].

Chapter 5

Implementation

In this Chapter, we present the variety of methods used to implement the ML Algorithms presented in Chapter 4. Firstly, we will discuss the necessary tools and libraries we have used to get the desired results. Furthermore, we demonstrate our proposed NDPF for the UNSW-NB15 dataset. The results gained from these implementations will be documented for performance analysis in Chapter 6.

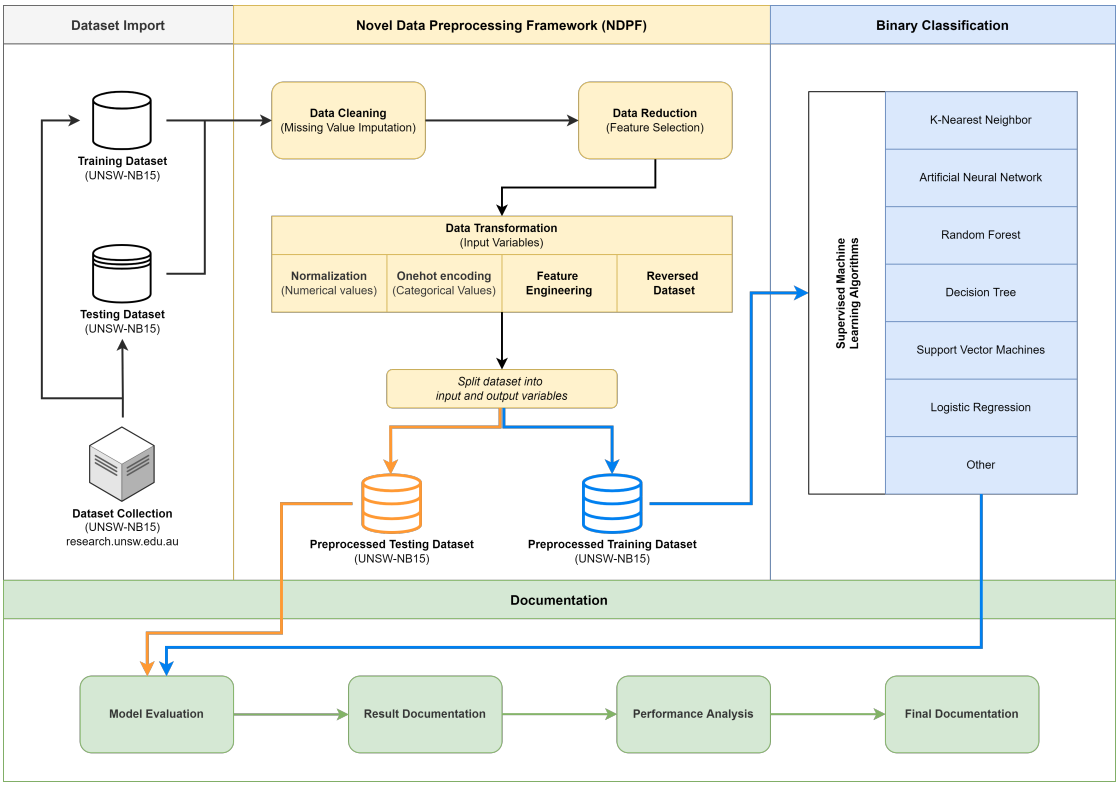


Figure 5.1: Project Framework for Anomaly Detection

Figure 5.1 shows the steps taken to implement our project framework. The first step of the whole project is *Dataset Import*, where the goal is to assess the chosen dataset. The UNSW-NB15 dataset can be acquired from UNSW Sydney's website[1] where the data is publicly open for any developer or researcher to access free of charge.

The second phase of the project is the development of our Novel Data Preprocessing Framework (NDPF), shown in Section 5.3. The RAW dataset undergoes several necessary preprocessing steps so that our ML models can apprehend and run the dataset without any errors. The main preprocessing steps include Data cleaning, Data Reduction, Data transformation, and splitting of the datasets into input and output variables.

The final two phases of the project focus simultaneously on ML algorithm development and implementation and documentation of the predictions. First, the training dataset is utilized for training the algorithms regarding Binary Classification, mentioned in Chapter 4, which include: K-Nearest Neighbor, Artificial Neural Network, Random Forest, Decision Tree, Support Vector Machines, and Logistic Regression. Finally, the testing sets validate the models, where the results are documented. The Result documentation is then analyzed for further inspection using various methods.

5.1 Tools and Python Libraries

This Section will present a variety of tools used for our thesis. The tools used are essential and provide different purposes to complete our project. Mainly, we use Python 3 as the programming language of choice. Even though Python is a slow interpreter language, compared to Java, C, and C++, the Python libraries' ease of use and availability are much more substantial. Figure 5.2 is a visual representation of our used tools and libraries.

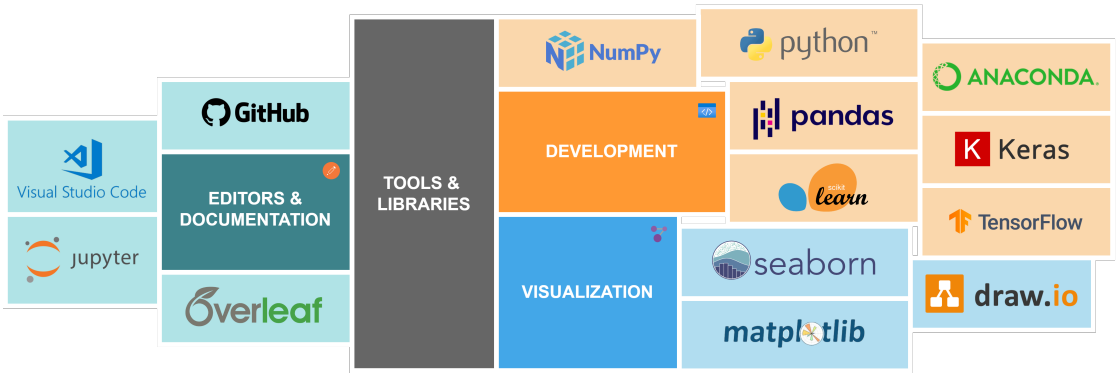


Figure 5.2: Tools and Libraries

5.1.1 Tools

This Section introduces the necessary tools for our thesis: *Python, Jupyter Notebook, Visual Studio Code, Github, Overleaf, Draw.io,* and *Anaconda.*

[1] `www.research.unsw.edu.au/projects/unsw-nb15-dataset`

Python

Python is a dynamically typed, interpreted language[2] with a concise and efficient syntax. Python's programming language is widely used and was created by Guido Van Rossum and published in 1991[113]. The programming language is frequently used to create websites and software, automate processes, and analyze data. Python is a general-purpose programming language, which means it can be used to develop a wide range of applications and is not tailored to any particular issue. It has become one of the most widely used programming languages because it is friendly for beginners and has versatility. According to a poll done by industry analytic company RedMonk, Python was the second-most famous programming languages[114] in 2021. There are many applications for Python:

1. Machine Learning and Data Analysis
2. Web application development
3. Automation of processes and scripting
4. Testing and prototyping software
5. Everyday tasks

In the world of Data Science, Python has become a pillar that allows data scientists and analysts to perform complicated statistical computations, produce data and visualize, develop and design ML algorithms, and have high flexibility within data manipulation. TensorFlow and Keras are two Python frameworks that let programmers create data analysis and ML systems more rapidly and effectively[115]. In addition, scientific computer languages such as R and Matlab are helpful for analysis, but they are challenging to incorporate into production applications. Unlike programming languages like C, C++[3], or Java[4], the time spent developing the code is minimal. As an outcome, developers may devote more effort to AI and ML algorithms and heuristics. In our case, Python 3 has been the only programming language we have used for the exact reasons mentioned above.

Jupyter Notebook

A Jupyter Notebook[5] is a free, open-source IDE that lets data scientists create and share documents with live code, equations, and other multimedia elements. We can use Jupyter Notebooks for Exploratory Data Analysis (EDA), data preprocessing (cleaning, transformation, visualization), modeling with statistics, ML, and deep learning.

Jupyter notebook are particularly excellent for presenting the work that has undergone. The notebook contains cells that can be run one after another for better understanding

[2] A programming language that executes the program, line by line
[3] `https://docs.microsoft.com/en-us/cpp/?view=msvc-170`
[4] `https://docs.oracle.com/en/java/`
[5] `https://jupyter.org/`

and structured code. The cells in the notebook can also be used as a markdown cell, where we can take notes and give helpful information to other developers[116], [117].

Visual Studio Code

Microsoft's Visual Studio Code (VS Code)[6] is a free, open-source text editor. Although the editor is moderately lightweight, it has powerful extensions that make it an excellent development environment. The platform supports a wide variety of programming languages: Python, Java, C++, CSS, and so on. In addition, it can identify whether any code snippets are missing. Variable definitions and standard variable syntaxes are also generated automatically. For example, if a variable is utilized in a program and the user forgets to define it, intelli-sense will declare it for the user. At the same time, numerous projects, including various files/folders, can be opened. These projects/folders might be connected or unrelated to one another. Resources may be retrieved and saved through the Git Hub Repo online repository. Resource pulling also refers to copying code made publicly available on the internet. This code can be modified and saved at a later time.

One of the main reasons for us choosing Visual Studio Code as a developing platform for our thesis is due to the extensions, especially Jupyter Notebook (`.ipynb` files). The editor made it easy for us to have multiple files open, reading .csv files at a glance and even restructure the file tree.

Github

Github is a cloud-based service that allows developers to store and manage code while tracking and controlling versions. Version control aids developers in tracking and managing changes to the code of a software project. Version control becomes increasingly important as a software project expands. We, as developers, can branch out sub-divisions of the project and make changes and modifications to a section of a code without jeopardizing the project as a whole. After branching, we can merge the section back into the main source code, which in hand can be published (or already published) after checking that the code is operating correctly. The central aspect of this process is to record the changes and backtrack if necessary[118], [119]. For our project, Github has been beneficial in tracking changes and sharing code files among us.

Overleaf

Overleaf provides a free online academic writing environment. The tool is used to create various documents from blog posts, journal articles, academic articles, posters, slide shows, books, essays, and so on. Although the system is based on LaTeX, it gives Overleaf a mature rich-text editor that one can use. LaTeX, is a documentation preparation system created by Leslie Lamport in 1986[120]. Overleaf allows real-time editing. For producing technical papers, LaTeX, facilities for typesetting mathematics make this an appealing choice. We have explicitly used Overleaf for documentation and writing for our thesis.

[6] `https://code.visualstudio.com/`

Draw.io

Draw.io[7] is a proprietary program for creating diagrams and charts created by Seibert Media[8]. We can use the software's automated layout option or design our own unique layout. They include a considerable number of shapes and hundreds of graphic components to let us create a one-of-a-kind illustration or chart. In addition, the drag-and-drop capability makes creating a professional-looking diagram or chart a breeze. Every chart and diagram that is not generated by Seaborn or Matplotlib (Section: 5.1.2) is drawn by us in Draw.io

Anaconda

Anaconda[9] is an open-source package manager focusing on data science for the Python and R programming languages. The purpose of Anaconda is to make managing packages (libraries) and deployment easier for the developer. Python environments are sensitive to versions across multiple libraries, where they have to be compatible with each other. Anaconda solves this challenge by an in-built control for versions before installing them to our Python Environment. Anaconda comes with over 250 in-built packages pre-installed, where over 7500 more packages can be installed from open-source databases. The Anaconda software is one of many data scientists' most used package managers.

5.1.2 Python Libraries

This Section introduces the necessary libraries for our thesis: *Pandas, Numpy, Matplotlib, Seaborn, Scikit-learn,* and *Keras.*

Pandas

The Pandas library is a powerful tool for working with data in the form of tabular data(excel, csv, SQL, JSON, ...), which may be presented in spreadsheets and databases. We can utilize the library to examine, clean and process data, where the dataset is often referred to as a `DataFrame` in Pandas[10].

The library is versatile since we can do several filtering methods for rows and columns or on a condition. Furthermore, there are several methods to slice, select, and extract the data which only we contradict necessary. In combination with the power of Matplotlib, Pandas can bring data into figures (plotting) with minimal effort. There are several plotting types: scatter, boxplot, barplot, and more.

We can apply simple statistics such as (mean, median, min, max, counts, etc). Those conditions can be used for the whole or sections of the dataset. The split-apply-combine method is another name for it. The library also offers strategies to manipulate textual data to apply ML models to our dataset. Table 5.1 represents the most used Pandas

[7] `https://drawio-app.com/`
[8] `https://info.seibert-media.net/display/DRAWIO/draw.io+Home`
[9] `https://www.anaconda.com/`
[10] `https://pandas.pydata.org/docs/pandas.pdf`

functions for our thesis[11]. In our project, the Pandas library has been an absolute necessity for data manipulation and preprocessing.

Numpy

Numpy (Numerical Python), created by Travis Oliphant[121] in 2005, is one of the most critical Python libraries, where its most common use is for Scientific Calculations on Python arrays. The open-source library also works in the domain of Fourier Transform, Matrices, and Linear Algebra. In Python, we work with arrays, which are extremely slow for processing, where Numpy has integrated an object of type `ndarray` which is up to 50 times faster than Python lists. Table 5.2, represents the most used Numpy functions[12].

Table 5.2: Most used Functions from Numpy library

Function	Description
`min()`	Find the Minimum value from a `ndarray`
`max()`	Find the Maximum value from a `ndarray`
`std()`	Find the Standard Deviation from a `ndarray`
`median()`	Find the Median from a `ndarray`
`percentile()`	Find the Percentile from a `ndarray`
`linspace()`	Create an evenly spaced range of values from an interval
`shape()`	Find the Shape of a `ndarray`
`reshape()`	Reshape a `ndarray`
`sort()`	Returns a sorted `ndarray`

Matplotlib & Seaborn

The graphic depiction of data is known as data visualization. It breaks down a large dataset into little graphs, making data analysis and prediction easier. It is a crucial component of data science that helps people comprehend and access complex data. Matplotlib[122] and Seaborn[123] are the foundations of Python data visualization.

Matplotlib is a Python package that helps plot graphs using other Python libraries like Numpy and Pandas. It is a robust Python data visualization tool. It is used to draw 2D graphs of arrays and create visually appealing figures. In addition, it makes use of Pyplot[13] to provide a free and open-source MATLAB[14]-like interface.

Seaborn is a Python package that works with Matplotlib, Pandas and Numpy to plot graphs and figures. We can consider Seaborn to be built upon the Matplotlib library to create univariate and bivariate data representations visually. Seaborn decorates Matplotlib visuals with beautiful, attractive graphics[124]. Table 5.3, represents a comparison of Seaborn and Matplotlib.

[11] `https://pandas.pydata.org/docs/reference/`
[12] `https://numpy.org/doc/`
[13] Pyplot is a set of functions that allow Matplotlib to behave similarly to MATLAB
[14] `https://www.mathworks.com/products/matlab.html`

Table 5.1: Most used Functions from Pandas library

Use case	**Function**	**Description**
Input/output	`read_csv()`	Read a .csv file and transform to a DataFrame
Input/output	`DataFrame.to_csv()`	Save a DataFrame as a .csv file to allocated path
Missing Data	`isna()`	Returns any missing values from DataFrame
Missing Data	`isnull()`	Returns any missing values from DataFrame
Missing Data	`notna()`	Returns any non-missing values from DataFrame
Missing Data	`notnull()`	Returns any non-missing values from DataFrame
Data Manipulation	`melt()`	Changes the DataFrame from wide to long.
Data Manipulation	`merge()`	Merges DataFrame
Data Manipulation	`get_dummies()`	Convert Categorical variables into dummy(indicator) variables.
Data Manipulation	`unique()`	Returns unique values from Dataframe
Index	`Index.values()`	Returns an array of the data in the feature/index
Index	`Index.is_unique()`	Returns if the index/feature has unique values
Index	`Index.has_duplicates()`	Returns if the index/feature has duplicate values
Index	`Index.hasnans()`	Returns TRUE if the index/feature has NAN
Index	`Index.dtype()`	Returns the data type for the data in the index/feature
Selecting	`Index.isin()`	Returns an array of booleans whether the given value corresponds to the index value.

Table 5.3: Matplotlib and Seaborn Comparison

Use case	Matplotlib	Seaborn
Functionality	Used to create simplistic graphs. Barplot, histplot, scatterplot, lineplot, and piecharts	Seaborn includes a number of data visualization patterns and visualizations. It employs intriguing themes. It facilitates the compilation of all data into a single graphic.
Visualization	Matplotlib is a graphical tool for data visualization in Python that works well with Numpy and Pandas. Pyplot has many of the same capabilities and syntax as MATLAB. Therefore, users of MATLAB can easily adapt this.	Seaborn is more experienced with Pandas data frames. In Python, it employs a simple collection of techniques to create stunning images.
Data Frames and Arrays	Matplotlib is good at working with data frames and arrays. Figures and axes are treated as objects. It has a number of domain specific plotting APIs. As a result, operations like plot() can operate without arguments.	Seaborn is far more structured and useful than Matplotlib, because it considers the entire dataset as a single entity.Because Seaborn is not very stateful, arguments are necessary when using methods like `plot()`

Scikit-Learn

Scikit-learn[125] is a popular ML toolkit for the programming language Python. Scikit-learn focuses on ML methods, statistical, mathematical, and various algorithms, which serve as the foundation for many ML technologies. In addition, Scikit-learn is a free tool that is extremely useful in constructing many different types of algorithms. David Cournapeau and Matthieu Brucher started the project as of Scikit-learn 2007, where the leadership for the project shifted to Fabian Pedregosa and others, and the first public release of the platform took place in 2011[125].

Classification, regression, and clustering techniques are the essential Scikit-learn features for ML. Scikit-learn, for example, allows work on RF, in which individual digital trees store node information aggregated in numerous tree topologies to produce a forest approach. Another way to put it is that each tree has clustered nodes in a tree topology, and the analysis from many trees is combined to create a global strategy that crunches data more precisely to reveal conclusions. In addition to RF, Scikit-learn can aid with gradient boosting, vector machines, and other ML features that are important for getting results. Scikit-learn, as a comprehensive resource, interacts with visualization and other tools like Matplotlib and Seaborn[126]. Listing 5.1, represents some of the imports from the Scikit-learn library.

```
1 # Importing Logistic Regression Classifier Library
2 from sklearn.linear_model import LogisticRegression
3
```

```
# Import Support Vector Machines Classifier Library
from sklearn.svm import SVC

# Import K-Nearest Neighbor Classifier Library
from sklearn.neighbors import KNeighborsClassifier

# Import Decision Tree Classifier Library
from sklearn.tree import DecisionTreeClassifier

# Import Random Forest Classifier Library
from sklearn.ensemble import RandomForestClassifier

# Import Multi Layer Perceptron (ANN) Classifier Library
from sklearn.neural_network import MLPClassifier
```

Listing 5.1: Import of classification libraries

Keras

Keras[15] is a Google-developed high-level deep learning API for creating neural networks. It is built in Python and is used to simplify neural network construction. It also allows for the calculation of numerous neural networks in the backend. Keras is simple to understand and use since it gives a high-level Python frontend with the flexibility of different backends for computation. Keras is slower than other deep learning frameworks because of this, yet it is incredibly user-friendly.

Keras lets us swap back and forth between several backends. Keras supports the following frameworks:

- Tensorflow[16]
- PlaidML[17]
- Theano[18]
- MXNet[19]
- CNTK(Microsoft Cognitive Tool)[20]

TensorFlow is one of the only five frameworks that has accepted Keras as its official high-level API. Keras is a deep learning framework built on TensorFlow and has integrated modules for all neural network operations. At the same time, we may use the Tensorflow Core API to create custom computations using tensors, computation graphs, sessions, and other things, giving us complete freedom and control over our application and allowing

[15] https://keras.io/
[16] https://www.tensorflow.org/
[17] https://www.intel.com/content/www/us/en/artificial-intelligence/plaidml.html
[18] https://theano-pymc.readthedocs.io/en/latest/
[19] https://mxnet.apache.org/versions/1.9.1/
[20] https://docs.microsoft.com/en-us/cognitive-toolkit/

us to execute our ideas in a short amount of time[127]. We have used Keras explicitly for developing our ANN Models.

5.2 The UNSW-NB15 Dataset

Dataset is the most crucial aspect of a network security analysis. The amount of the data also impacts the performance of ML algorithms. Therefore, selecting the most significant features from the input data can simplify the modeling process and result in faster and more accurate predictions. Frequently, datasets contain multiple useless, redundant features, which is disadvantageous to the accuracy of the results. In light of this, the UNSW-NB15 dataset is selected. The dataset has also been widely used in many types of research.

The UNSW-NB15 dataset was created by researchers from the Australian Center for Cyber Security (ACCS) in 2015 with 49 features and nine attack categories[17]. The UNSW-NB15 dataset is more sophisticated and reflective of current attacks and regular network traffic, making it suitable for evaluating network intrusion techniques. The dataset unprocessed (RAW) network packages were generated using the IXIA PerfectStorm system[21] in UNSW Canberra's Cyber Range Lab. The goal was to produce a hybrid of genuine modern regular activities and synthetic recent attack behaviors. Whereas 100 GB of RAW traffic was captured using the tcpdump program (Pcap files)[17]. The datasets are allocated in a *cloudstor* database, where the data are categorized into folders: *Argus, BRO, CSV, PCAP*, and *Reports*. Argus is a network and system monitoring tool. Its purpose is to keep track of network services, servers, and other network equipment. The BRO is an anomaly-based Intrusion Detection System (IDS), where the files for this particular system are located in a folder correspondingly. The CSV (Comma Separated Values) folder is the data our project mainly focuses on, where the dataset providers offers already split training and testing sets[31], [128].

5.2.1 Attack Categories

The dataset includes nine different attack categories: Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode, and Worms[1], [129].

1. **Fuzzers** is a method for adversaries (hackers) to automatically crash a system or program by injecting randomly generated data.

2. **Exploits** takes advantage of security vulnerabilities in the system or network. As a result, the adversary knows a problem within a specific application and takes leverage with that knowledge.

[21] The PerfectStorm module from xia is a scaleable solution for testing convergent multi-play services, application delivery systems, and network security platforms on both wired and wireless networks (`www.keysight.com/us/en/products/network-test/network-test-hardware/perfectstorm.html`)

3. **Denial of Service (DoS)** attack shuts down networks and machines and denies access to the original owner or user. Denial of Service attacks is carried out by overwhelming the target computer of the network with information until it crashes. Victims of Denial of Service attacks are often high-profile private and public targets, such as businesses, banks, and companies within the media, resulting in the victim using vast sums of money to fix the damage.

4. **Worm** is a harmful software that continuously duplicates itself and spreads through a network to other computers. Worm virus exploits flaws within the security software to steal crucial data, establish backdoors that may be exploited to connect to the network, damage data, and perform other types of abuse.

5. **Reconnaissance** refers to information gathering by unwanted adversaries. The approach for this attack is either remotely or on-site, using various techniques as well as *Social Engineering*. Some of the most common Reconnaissance attacks are gathering data about packets (sniffing), ping sweeping, port scanning, and phishing.

6. **Shellcode** is when adversaries transmit code instructions to an arbitrary software with malicious intent to execute hostile activities.

The mentioned network attack types are merely a tiny fraction of malicious network activities, where the possibilities are endless for one with harmful intentions.

5.2.2 Structure

The number of records in the UNSW-NB15 dataset is $2,540,044$ which is stored in four `.csv` files represented in Table 5.4. Each of the datasets contains attack and normal records. A training set and a testing set were created from the data collection, named UNSW-NB15_training-set.csv and UNSW-NB15_testing-set.csv, respectively. The training set has $82,332$ records, while the testing set contains $175,341$ records, including attack and normal records. Do note that the training and testing sets are reversed. Finally, Table 5.5 represents the distinct features included in the UNSW-NB15 dataset, and their data types.

Table 5.4: UNSW-NB15 files

File name	File Size	Records	Features
UNSWNB15_1.csv	161.2 MB	700000	49
UNSWNB15_2.csv	157.6 MB	700000	49
UNSWNB15_3.csv	147.4 MB	700000	49
UNSWNB15_4.csv	91.3 MB	440044	49
UNSWNB15_testing-set.csv	31.5 MB	175341	45
UNSWNB15_training-set.csv	15.0 MB	82332	45

Table 5.5: UNSW-NB15 List of Features

Feature Number	**Feature Name**	**Data Type**	**Feature Number**	**Feature Name**	**Data Type**
f1	srcip	nominal	f26	res_bdy_len	integer
f2	sport	integer	f27	Sjit	Float
f3	dstip	nominal	f28	Djit	Float
f4	dsport	integer	f29	Stime	Timestamp
f5	proto	nominal	f30	Ltime	Timestamp
f6	state	nominal	f31	Sintpkt	Float
f7	dur	Float	f32	Dintpkt	Float
f8	sbytes	Integer	f33	tcprtt	Float
f9	dbytes	Integer	f34	synack	Float
f10	sttl	Integer	f35	ackdat	Float
f11	dttl	Integer	f36	is_sm_ips_ports	Binary
f12	sloss	Integer	f37	ct_state_ttl	Integer
f13	dloss	Integer	f38	ct_flw_http_mthd	Integer
f14	service	nominal	f39	is_ftp_login	Binary
f15	Sload	Float	f40	ct_ftp_cmd	integer
f16	Dload	Float	f41	ct_srv_src	integer
f17	Spkts	integer	f42	ct_srv_dst	integer
f18	Dpkts	integer	f43	ct_dst_ltm	integer
f19	swin	integer	f44	ct_src_ ltm	integer
f20	dwin	integer	f45	ct_src_dport_ltm	integer
f21	stcpb	integer	f46	ct_dst_sport_ltm	integer
f22	dtcpb	integer	f47	ct_dst_src_ltm	integer
f23	smeansz	integer	f48	attack_cat	nominal
f24	dmeansz	integer	f49	Label	binary
f25	trans_depth	integer			

Before performing data preprocessing, we wanted to observe how balanced the UNSW-NB15 dataset was. Figure 5.3 shows that the distribution between attack labels and normal (non-attack) labels is slightly imbalanced, this problem is known as *Class Imbalance*. To be precise, the attack labels make up 63.907% of the total number of labels, whereas the normal labels make up 36.092%. The slight imbalance would not significantly impact our models and performance evaluation, where we can confirm whether our results are valid with the help of several performance metrics.

5.3 Novel Data preprocessing Framework (NDPF)

Before any ML algorithm is applied to a dataset, the data has to be preprocessed so that the processed data is in the format that the model can comprehend. RAW data is obtained from a domain contextually to address an issue and propose a solution. Therefore, RAW

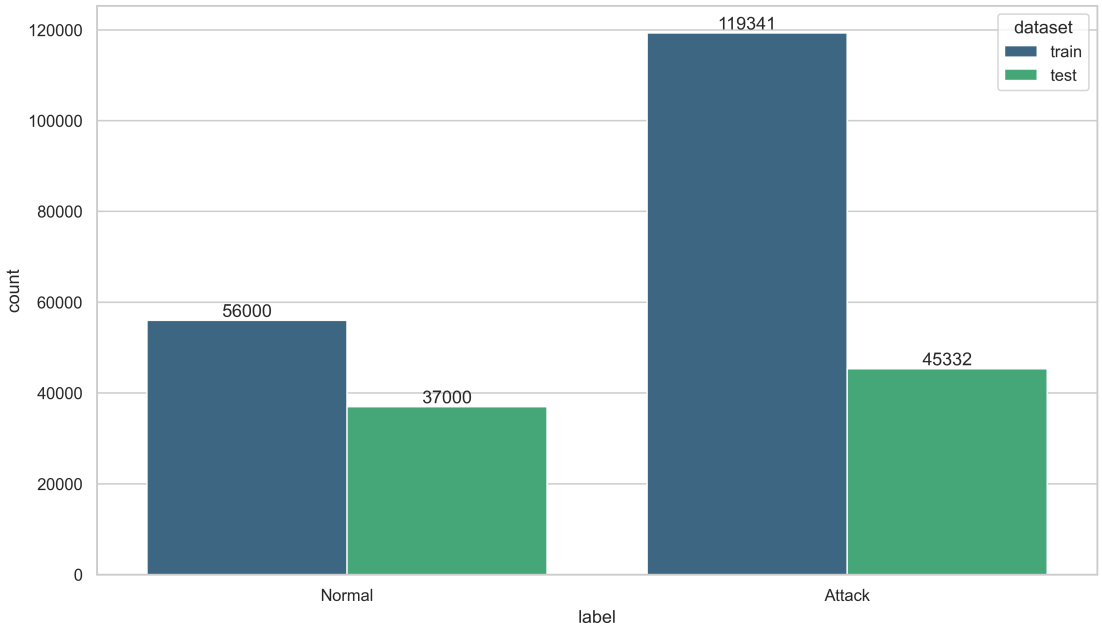


Figure 5.3: Normal and attacks distributed

data must virtually always be transformed before being used as a foundation for ML models (Figure 5.5). Hence, our goal is to create a Novel Data Preprocessing Framework (NDPF), to preprocess and transform the UNSW-NB15 dataset. Figure 5.4 shows the usual time spent when working on ML projects, estimated by us.

The initial stage in our NDPF is to acquire the dataset before applying it to any model. This dataset will consist of data acquired from various sources and merged to make a dataset. A dataset can also be generated by using several Python APIs to collect information, where we can finally save the dataset in one of many formats: CSV, HTML, or XLSX[130]

The data from the domain is most likely organized in a single big table with rows and columns, where we have to consider that each column/feature inherits multiple data types. For example, the numerical variables include floating points, integers, ranks, percentages, and currency, whereas categorical variables include names, categories, and labels which consist of letters, signs, and words. The final form of variable could be binary, which includes 0 and 1, being `TRUE` and `FALSE`. The challenge is that ML algorithms are based on numerical data, meaning they only accept, in most cases, numerical values representing vectors and matrices from Linear Algebra[42].

The results can still vary even though our NDPF fulfills each model's requirement, and even when an algorithm's assumptions are disregarded or entirely broken, it is normal for it to perform as well as or more significant than other techniques. This is a typical occurrence that must be considered while developing and evaluating ML algorithms. Therefore, exemplary quality data is essential for training ML models. Conversely, data

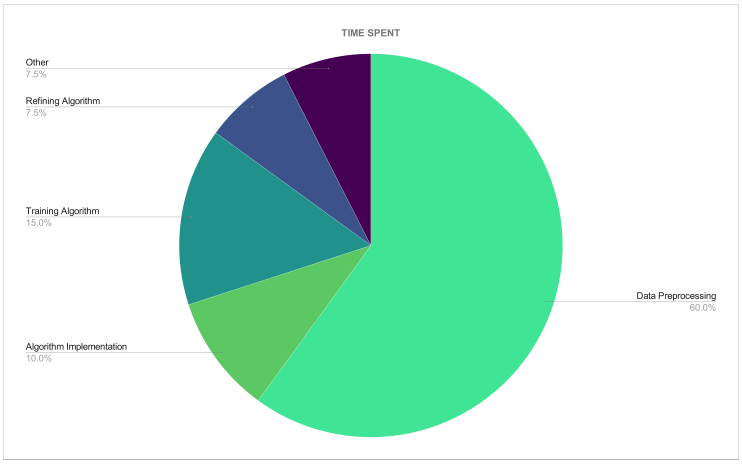


Figure 5.4: Time Spent

with poor quality might refer to a "poor representation" of the problem that has to be solved, where the model will fail to convey the necessary dynamics for learning how to translate instances of inputs to outputs[42].

1. **Complex Data:** Raw data often includes condensed, complicated nonlinear connections that must be uncovered.

2. **Messy Data:** Raw data comprises statistical noise, mistakes, missing numbers, and instances that contradict each other.

To summarize, data preparation leads to better data and, as a result, greater model performance.

The UNSW-NB15 contains categorical and non-similar scale features that must be preprocessed to make the dataset compliant with a ML algorithm. To ensure that the selected dataset is converted into a suitable one, we develop our NDPF with *data cleaning*, *data transformation*, and *data reduction*, as shown in Figure 5.5.

1. **Data Cleaning:** aims to process missing and irrelevant data-points.

2. **Data Transformation:** aims to convert data into a readable format for a given ML model.

3. **Data reduction:** aims to narrow down the dimensionality of the dataset to improve model efficiency and accuracy.

Figure 5.5: Preprocessing Stages

5.3.1 Data Cleaning

The first phase of our NDPF is Data Cleaning, which aims to fix the problem with incomplete, inconsistent, corrupt, duplicate, and noisy data.

However, first, missing or irrelevant data needs to be processed. *Missing data points* is a common attribute that many datasets have. Even if the data looks valid, results and algorithms are untrustworthy if the data is inaccurate. Redundant observations are most likely to occur during the data gathering process. We can address the missing values in Python by using Listing 5.2. There are several approaches to solving this problem.

1. **Dropping duplicate values:** Dropping a duplicate row or column and keeping the first instance is one of the solutions to this problem. The reason for doing this is not to make the specific data point take advantage of or be biased.

2. **Dropping missing values:** Dropping rows and columns that includes a not a number (`NaN`) field or an empty object (`NULL`) values.

3. **Data imputation:** Estimating missing values is used if a small percentage of the data is lost. Data imputation techniques such as mean, median, or mode values are the most common approach to apprehend this problem.

```
# To check if train and test datasets inhibits missing values
train.isnull().sum()
test.isnull().sum()
```

Listing 5.2: `isnull().sum()` function

After a thorough analysis, the UNSW-NB15 does not exhibit any missing values to remove or perform data imputation.

5.3.2 Data Transformation

The second phase our NDPF is Data Transformation, which converts the data into a known format readable to a ML model. Data normalization transforms numerical

columns into a standard scale (−1.0 to 1.0 or 0.0 to 1.0). This process transforms the data columns without distorting the differences in the data range, meaning it will maintain the dynamic range of the dataset, which makes it easier for the algorithm to determine the relative link between data points.

Reversed Datasets

The UNSW-NB15 dataset has the training and testing set samples reversed (Listing: 5.3), caused by an error from the publishers. This means that the testing set samples are double the training set size. We promptly fix the issue by running an `if` statement on both sets and checking whether the training set has fewer or larger samples than the testing set. As a result, the testing set contains 175, 341 rows and 45 columns instead of the training set, corresponding to 82, 332 rows and 45 columns.

```
if train.shape < test.shape:
    # Reversing the dataset
    train, test = test, train
else:
    print("The dataset, is already reversed")
return train, test
```

Listing 5.3: Reversing the datasets

Feature Engineering

The process of changing the RAW data into features that better describe the underlying problem to predictive models, resulting in enhanced model accuracy on unseen data, is known as feature engineering. Feature engineering depends on domain knowledge in a specific field to select and transform features from a dataset. It is evident that many features can share requirements characteristics. The procedure entails a combination of data analysis, rule-of-thumb use, and judgment.

The feature engineering process starts with analyzing the various values for each feature for the dataset. Then, by executing `train['state'].value_counts`[22] from the pandas library, we output the row count for each distinct value in a feature, in this case, 'state'. Table 5.6 represents the row counts for the states 'FIN', 'INT', 'CON', and 'REQ' are more significant than the others. Therefore, the states with lower row counts have been renamed 'Others'. This process is called binning.

[22] `https://pandas.pydata.org/docs/reference/api/pandas.Series.value_counts.html`

Table 5.6: Number of Instances (State)

State	Count	State	Count
INT	82275	ECO	12
FIN	77825	PAR	1
CON	13152	URN	1
REQ	1991	NO	1
RST	83		

The '-' value is present under the 'service' feature, which equals 47, 153. This value is converted to 'Others' since the dataset description informs us that the value is a service that is not frequently used. The same method is replicated for the 'service' and 'proto' features. We also observed the row counts for those features and renamed the values to 'Others'. The purpose of this procedure is to reduce the complexity of the model. Table 5.7 represents the row counts for the distinct values of the 'services' column.

Table 5.7: Number of Instances (Services)

Services	Count	Service	Count
-	47153	ssh	204
dns	21367	ssl	30
http	8287	snmp	29
smtp	1851	dhcp	26
ftp	1552	radius	9
ftp-data	1396	irc	5
pop3	423		

Categorical Features

Our ML algorithm cannot process Categorical data points, meaning the transformation of the data points into a readable format is essential for the model to run without errors. Therefore, we drop the 'attack_cat' feature since our model performs Binary Classification. We address the different data types for each feature, by implementing Listing 5.4 in Python.

```
# Addressing the different Data types for each column
train.dtypes
test.dtypes
```

Listing 5.4: `dtype` method for checking data types

Many data science tools can only function with numerical values as input, which is especially true in terms of ML. For example, if we try to input string-based category (`dtype object`) data, the designated model will output an error. Hence, to make the categorical features readable by the model, one-hot encoding needs to be implemented.

As a result, we transform each categorical value into a new categorical column (dummy indicator variables) and assign a binary value of 1 or 0 to the columns using one-hot encoding, as shown in Figure: 5.6. A binary vector is used to represent each integer value. The `pd_dummies()`[23] function from the Pandas library accomplished exactly this assignment, as presented in Listing 5.5.

```
# Using get_dummies to make the categorical values usable.
x_train = pd.get_dummies(x_train)
x_test = pd.get_dummies(x_test)
```

Listing 5.5: `pd.get_dummies()` function

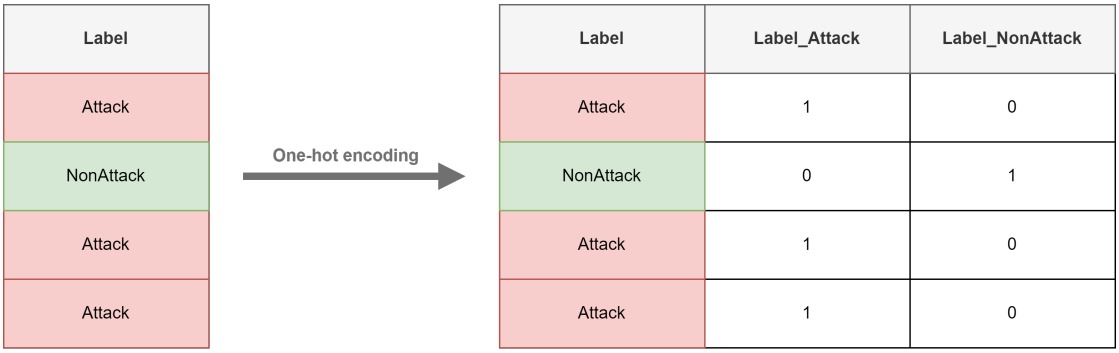


Figure 5.6: One-hot-encoding (OHE)

Normalization

In ML, normalization is a method for preparing data. Normalizing is converting the values of each numerical feature to the same scale. But, the technique is not necessary for every dataset, only for the dataset where the values of the feature ranges are significant. Nevertheless, normalization and standardization help several ML methods, primarily when Euclidean distance (KNN Section: 4.6) is utilized. For instance, say one of the KNN variables is in the 1000s, while another variable is 0.1s, the highest value will heavily influence the distance calculation. Therefore, normalization and standardization may be advantageous in this situation[131].

The UNSW-NB15 dataset has a broad dynamic range of values, indicating that normalization needs to be applied to the numerical columns. The core concept of standardizing/normalizing is to individually delegate $\mu = 0$, and $\sigma = 1$ for the features of X. `Standardscaler()`[24] will independently normalize the features so that each column/feature will produce $\mu = 0$, and $\sigma = 1$, as mentioned. Python implementation for `StandardScaler()` is presented in Listing 5.6.

[23] `https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html`

[24] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`

x equals the observation, μ is the mean, and σ is the standard deviation. The Equation for standard deviation (Equation: 5.3) is achievable by taking the equation for standardization (Equation: 5.1) and combining it with the mean for each sample (Equation: 5.2).

$$z = \frac{x - \mu}{\sigma} \tag{5.1}$$

$$\mu = \frac{1}{N}\sum_{i=1}^{N}(x_i) \tag{5.2}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{5.3}$$

```
# Using standard scaler to normalize data on non categorical columns
scaler = StandardScaler()
x_train[non_cat_cols] = scaler.fit_transform(x_train[non_cat_cols])
x_test[non_cat_cols] = scaler.transform(x_test[non_cat_cols])
```

Listing 5.6: `StandardScaler()` function

5.3.3 Data/Dimensionality Reduction

Dealing with a large amount of data for analysis comes with difficulty. The dimensionality of a dataset refers to the number of features or input variables. The 'Curse of Dimensionality' refers to how adding more input data makes predicting modeling more challenging to apprehend. Hence, data reduction (dimensionality reduction) improves storage efficiency and lowers data storage and analysis expenses. The concept of data reduction is to decrease the number of features or dimensionality. Data reduction is, in our case, done in Feature Selection. Data visualization frequently employs high-dimensionality statistics and dimensionality reduction methods. However, in applied ML, the same approaches may be used to reduce a classification or regression dataset to better train a prediction model[132]. Firstly, we analyze the dataset by looking at the correlation matrix.

Correlation

Correlation is the degree to which two variables are linearly related. If there is an extensive dataset with many columns, displaying the correlation matrix as a heatmap is a rapid approach to assess relationships between columns. When a dependent and independent variable has a high correlation value, the independent variable affects the final output.

- -1 implies a perfect negative linear correlation between two variables.
- 0 implies that there is no linear relationship between two variables

- 1 implies a perfect positive linear correlation between two variables.

The Pearson r correlation is the most extensively utilized statistical approach for determining the degree of linearly related variables. The Pearson correlation between any two variables x, y can be calculated by using Equation 5.4, where n is number of observations, and i-denotes the i-th observation.

$$r_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}} \tag{5.4}$$

Figure 5.7 shows a graphical representation of the correlation matrix for several features with coefficient values given by a color scale.

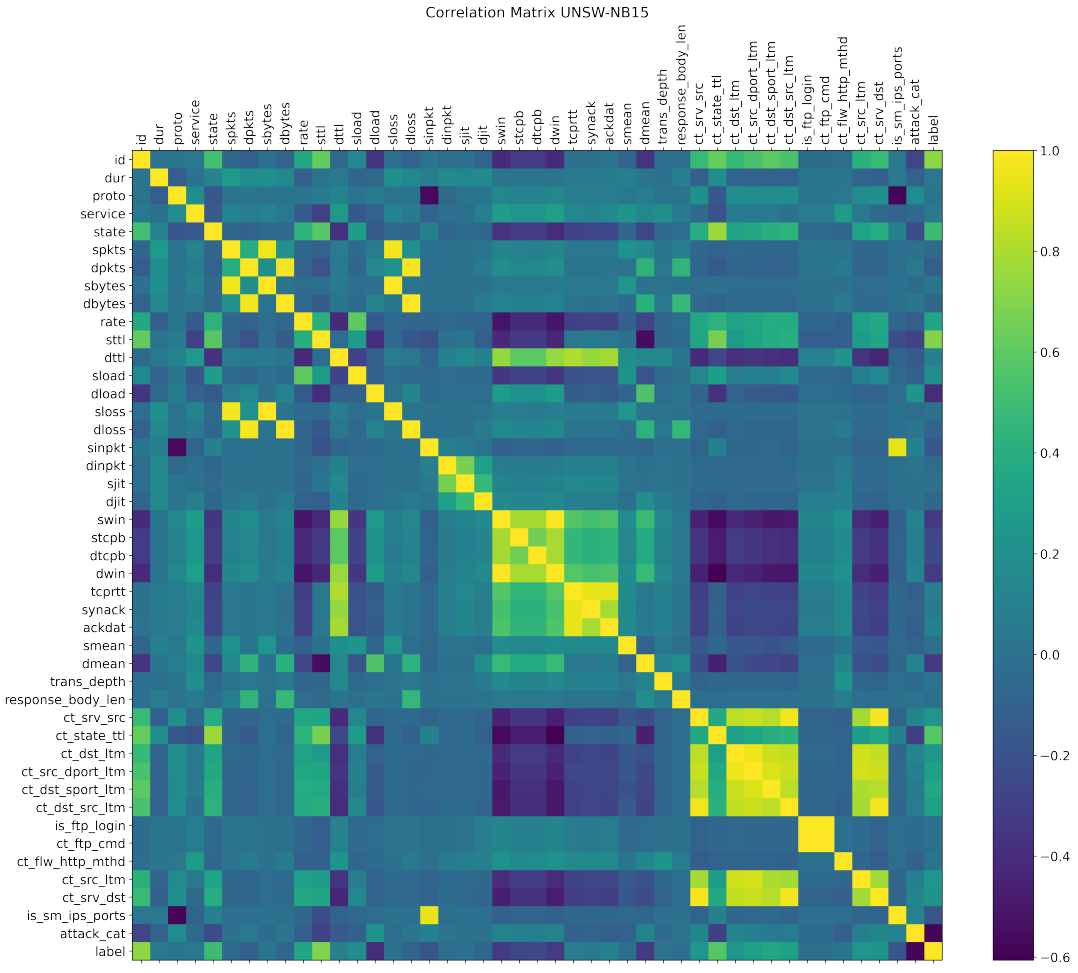


Figure 5.7: Correlation plot for UNSW-NB15 Training dataset

Feature Selection

Feature selection is one of the core concepts in ML. This process hugely impacts the performance of any ML model. When developing a predictive model, the primary objective is to reduce the number of input variables. The number of input variables included in a model's training stage dramatically influences the performance, meaning feature selection and data cleaning should be the first and most crucial step of the developing phase.

Reducing the number of input variables lessens the computational cost and, in many cases, enhances the performance of a given model. The process has two given approaches:

1. Automatically determine input variables using various algorithms

2. Manually choose input variables to remove from the model.

The benefits of reducing the number of input variables are:

1. Reduce overfitting i.e, the dataset will become less redundant, which in hand will make fewer opportunities to make a decision based on noise,

2. Accuracy improvement i.e, training data will be less misleading, which will improve accuracy

3. Training Time reduction i.e, the complexity of the algorithm decreases.

It is best to pick features by analyzing just the training set in all feature selection operations. This prevents overfitting.

Method 1 - Recursive Feature Elimination (RFE)
Recursive Feature Elimination (RFE) is a well-known feature selection algorithm used to identify which features in a training dataset are more significant in predicting the target variable. RFE works by searching for a subset of features in the training dataset, starting with all features and successfully eliminating them until the targeted number remains. The method can be accomplished by re-fitting the model using the provided ML technique, ranking features by relevance, excluding the least important features, and fitting the model again. This procedure is executed until only a certain amount of features are left. Features are evaluated by using either a ML model or a statistical technique. In our work, RFE was executed with Cross-validation (CV). As a result, the algorithm has concluded with the top features and has left us with the following features to drop: `is_sm_ips_ports`, `dwin`, `ct_ftp_cmd`, `is_ftp_login`, where the results from the algorithm are represented in Table 5.8. RFE is described in Algorithm 4[133]

Algorithm 4: Recursive Feature Elimination (RFE)

Data: Training Data $D = \{f_1, f_2 \dots f_n\}$
Result: Feature Ranking $R = r_{f_1}, r_{f_2} \dots r_{f_n}$
for *each feature* f_i *in* D **do**
    $D' = D - f_i$
    Train the classifier with D'
    Compute the classification accuracy
    Determine the accuracy loss of the classifier, after eliminating f_i
end
Calculate the loss for every feature $\{f_1, f_2 \dots f_n\}$
Compute the rank of each feature $R = r_{f_1}, r_{f_2} \dots r_{f_n}$, with the loss in consideration

Table 5.8: Results: Recursive Feature Elimination with Cross-validation (RFECV)

Feature	Support	Rank	Grid_score	Feature	Support	Rank	Grid_score
dur	True	1	0.82251	proto	True	1	0.8662
dtcpb	True	1	0.95708	service	True	1	0.87824
tcprtt	True	1	0.9573	state	True	1	0.89052
synack	True	1	0.95798	spkts	True	1	0.89932
ackdat	True	1	0.95903	dpkts	True	1	0.93425
smean	True	1	0.95907	sbytes	True	1	0.93292
dmean	True	1	0.95972	dbytes	True	1	0.94531
trans_depth	True	1	0.96004	djit	True	1	0.95715
ct_srv_dst	True	1	0.96196	rate	True	1	0.94964
response_body_len	True	1	0.96054	dttl	True	1	0.95658
ct_state_ttl	True	1	0.96087	sload	True	1	0.95606
ct_dst_ltm	True	1	0.96057	dload	True	1	0.95691
ct_src_dport_ltm	True	1	0.9613	sloss	True	1	0.95532
ct_dst_sport_ltm	True	1	0.96089	dloss	True	1	0.95753
ct_dst_src_ltm	True	1	0.96168	sinpkt	True	1	0.95808
ct_flw_http_mthd	True	1	0.96183	dinpkt	True	1	0.95807
ct_src_ltm	True	1	0.962	sttl	True	1	0.95513
ct_srv_src	True	1	0.96083	is_sm_ips_ports	False	2	0.96201
swin	True	1	0.95745	dwin	False	3	0.95742
stcpb	True	1	0.95677	ct_ftp_cmd	False	4	0.96231
sjit	True	1	0.95734	is_ftp_login	False	5	0.96162

Method 2 - Random Forest Classifier with Cross-validation
After RFE, we implemented RF Classifier to select our features, which resulted in more significant results. RF feature selection falls under the area of Embedded techniques, which are highly accurate, generalize better, and are interpretable. Filter and wrapper methods are combined in embedded methods. Algorithms with built-in feature selection techniques are used to implement them.

If the task is classification, Gini impurity or information entropy/gain are used to classify the impurities. Then, the impurity for regression is calculated by using the

variance. As a result, it is easy to figure out how much each feature reduces impurity while training a tree. Of course, the feature that decreases the impurity most is the most valuable. The impurity decrease from each feature in a RF may be averaged across trees to establish the variable's final significance. To better understand, features picked at the top of the trees are typically more essential than those at the bottom of the trees, as top splits generally result in more considerable information gains[134].

Our method using RF Classifier as the foundation included six steps:

1. Importing necessary libraries.
2. Defining custom functions.
3. Preparation of data
4. Running Random Forest Classifier for training data (and 10-fold cross-validation)
5. Running Random Forest Classifier for testing and training data (and 10-fold cross-validation).
6. Save the output.

The process started with importing necessary libraries, including NumPy[25], Pandas[26], Sci-kit learn (label encoder[27], StandardScaler[28], metrics[29], StratifiedKFold[30], RandomForestClassifier[31]), and tqdm notebook[32].

The second step of the process utilized one of our functions to import the training and testing dataset while dropping "attack_id" and "id." Furthermore, we tend to inspect whether the dataset is reversed or not by running an `if` statement. Finally, we also utilized a function that appended all categorical columns (Listing 5.7) into an empty list. We also initialize the `StratifiedKFold()` with parameter tuning, where `n_splits= 10`, `seed = 1`, and `shuffle = True`. Finally, for step two, we create a dictionary "`importance_dict`" where all of the features from the dataset are listed.

The final step executes `RandomForestClassifier()` for a total of four times, where two of them include 10-fold Cross-validation. Firstly we apply the algorithm on the training dataset two times, where one with 10-fold Cross-Validation applied, then append it to the `importance_dict`, as shown in Listing 5.8. We conduct the same process for the training

[25] www.numpy.org/doc/

[26] www.pandas.pydata.org/pandas-docs/stable/

[27] www.scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

[28] www.scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html?highlight=standardscaler#sklearn.preprocessing.StandardScaler

[29] www.scikit-learn.org/stable/modules/classes.html?highlight=metrics#module-sklearn.metrics

[30] www.scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html

[31] www.scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[32] www.tqdm.github.io/docs/notebook/

and testing datasets combined, for further inspection and analysis. Finally calculate the mean value for from all the investigations, together. Figure 5.8, and 5.9, represents the density and distribution of the results.

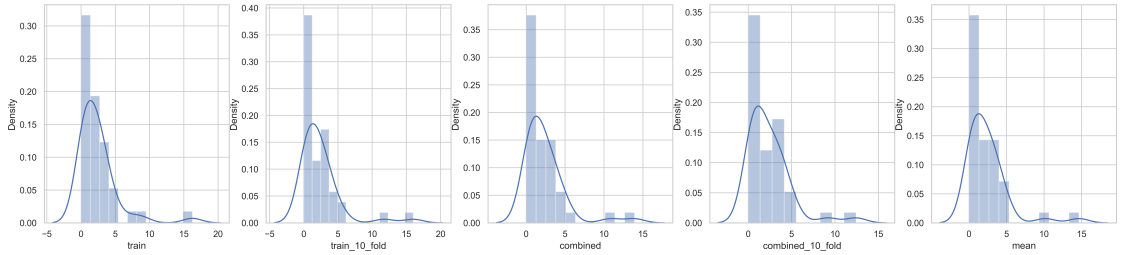


Figure 5.8: Distribution Plot: Feature Selection with `RandomForestClassifier()`

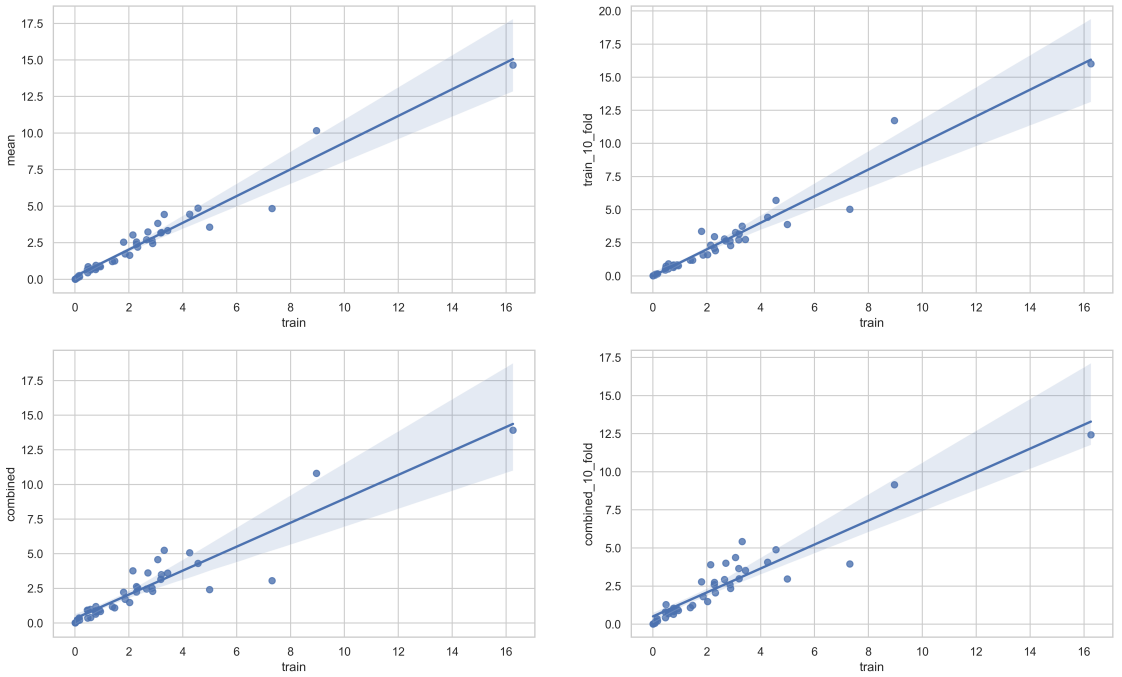


Figure 5.9: Regression Plot: Feature Selection with `RandomForestClassifier()`

```
def get_categorical_columns(train):
    categorical_columns = []
    for col in train.columns:
        if train[col].dtype == 'object':
            categorical_columns.append(col)
    return categorical_columns
```

Listing 5.7: Function that returns all categorical columns in dataset

```
# Initializing RandomForestClassifier
clf = RandomForestClassifier(random_state=1)
# Fitting x_train and y_train
clf.fit(X, Y)
feature_importance = clf.feature_importances_
# Adding the values into a new column 'train'
importance_dict['train'] =  feature_importance
```

Listing 5.8: `RandomForestClassifier() on training set`

By analyzing our results from Table 5.9, we conclude with dropping the following features: `response_body_len`, `spkts`, `ct_flw_http_mthd`, `trans_depth`, `dwin`, `ct_ftp_cmd`, and `is_ftp_login`. In addition we, drop `ID` since it does not contribute to any additional valuable information. We also tend to drop `attack_cat`, since our work mainly focuses on Binary Classification.

5.4 Algorithm Implementation

Not every algorithm is suitable for every dataset in ML. Even though one algorithm may excel in performance over another, it might not be the most appropriate in a real-world scenario. Therefore, in our thesis, we have implemented six different algorithms, with different parameter combinations to get a comprehension of the overall performance evaluation to conclude, which algorithm performs best on our preprocessed dataset. Every algorithm we have employed is thoroughly explained in Chapter 4. We chose those particular algorithms since they are well-known and strong supervised learning algorithms. Furthermore, other researchers use them on the same dataset, making it easier to compare and evaluate.

There are many limitations with ML projects regarding how computational heavy many of the algorithms are and how time-consuming they can become, especially when running hyperparameter tuning. This was an essential factor when considering what parameters and how big the grid would be for hyperparameter tuning. For example, several algorithms took as long as eight days to export our desired results in our thesis, even though the CPU were operating with all available cores.

The theory behind every ML algorithm implemented is necessary knowledge. Therefore, it is essential to understand how the algorithms work and how their parameters affect the model outcome. For example, using the knowledge of how parameters work for DT and RF helps us define reasonable parameter ranges. Prior knowledge of the parameters and libraries can help choose parameter values closer to correct. In addition, reading the documentation for each algorithm and knowing the default parameters can help mitigate algorithm runtime significantly and even understand to avoid overfitting and underfitting.

At the beginning phase of the thesis, we had some underlying statements and assumptions that Neural Networks would be the algorithm that performs best, both with and without tuning, as it is a robust algorithm. The reason is that ANN can learn highly complex relations and non-linear data, which is essential when looking at real-life data

Table 5.9: Feature Selection with `RandomForestClassifier()` with 10-fold Cross-validation

Feature	**Train**	**Train (10-fold)**	**Train + Test**	**Train + Test (10-fold)**	**Mean**
sttl	16.2550945	15.0421223	10.9670924	13.3090943	13.8933509
ct_state_ttl	8.9619085	11.5963044	11.2640431	9.03038	10.213159
dload	4.5674637	6.7575214	6.3604314	5.7830982	5.8671287
dttl	7.3116212	4.9804259	3.8173527	3.6431592	4.9381398
rate	4.253581	4.5679322	5.0098576	4.5295328	4.5902259
dmean	4.9919352	4.3480433	3.2015804	2.6198709	3.7903574
sload	3.310175	4.232429	4.4274681	5.2797084	4.3124451
ackdat	1.8057622	3.3503576	2.5293968	2.5270787	2.5531488
sbytes	3.0677247	3.1999918	4.4615967	4.2624629	3.747944
tcprtt	3.2027874	2.7853994	2.2972975	2.6713096	2.7391985
dbytes	3.1836302	2.7355933	3.4009155	3.5361747	3.2140784
ct_srv_dst	3.4354459	2.7254739	3.7892882	3.5091879	3.3648489
smean	2.7068465	2.679014	4.1435376	3.9979471	3.3818363
synack	2.6543144	2.6709122	3.0291934	2.7321854	2.7716513
dur	2.8625865	2.5687961	1.6937501	2.6213817	2.4366286
dinpkt	2.284652	2.556445	2.9892422	2.706975	2.6343286
ct_dst_src_ltm	2.1452773	2.3582964	3.5216035	3.8578813	2.9707646
sinpkt	2.87987	2.2483162	1.7933931	2.3789027	2.3251205
ct_srv_src	2.2793012	2.1567363	2.6123842	2.5902471	2.4096672
sjit	2.0277716	1.6794793	1.9847377	1.4587462	1.7876837
dpkts	2.32241	1.5607976	2.0820009	1.9278532	1.9732654
state	1.8591436	1.4902598	1.4022439	1.6092294	1.5902192
djit	1.4709708	1.2658864	1.4040854	1.3296465	1.3676473
proto	1.3879482	1.1960054	1.282419	1.1212633	1.246909
ct_dst_sport_ltm	0.4876	0.9264636	0.8568151	1.456689	0.9318919
ct_src_ltm	0.9419657	0.8391595	0.9396028	0.8849734	0.9014253
sloss	0.771581	0.8139839	1.0211508	1.1551714	0.9404718
dloss	0.5812025	0.8081575	0.5817343	0.6356618	0.651689
ct_dst_ltm	0.7757124	0.7965062	0.854861	0.869824	0.8242259
dtcpb	0.7774476	0.7763523	0.985237	0.8448853	0.8459806
spkts	0.9049584	0.7596935	1.1612101	0.989895	0.9539393
stcpb	0.7602875	0.7487856	0.7873531	0.8709101	0.7918341
ct_src_dport_ltm	0.7642101	0.612859	0.6188303	0.6197144	0.6539034
is_sm_ips_ports	0.4688463	0.5994069	0.397046	0.3685598	0.4584648
swin	0.5698937	0.5640329	0.5702751	0.6874545	0.597914
service	0.456995	0.4701579	0.8793855	0.8146467	0.6552963
response_body_len	0.1542974	0.1706953	0.4137625	0.3367497	0.2688762
ct_flw_http_mthd	0.1703705	0.1573909	0.2392378	0.2111502	0.1945374
trans_depth	0.0927764	0.1241565	0.1221516	0.144337	0.1208554
dwin	0.0757839	0.05687	0.0864489	0.0530457	0.0680371
is_ftp_login	0.0103945	0.0115616	0.0110969	0.010804	0.0109642
ct_ftp_cmd	0.0074555	0.0112277	0.0088904	0.0122116	0.0099463

as most of the actual data is not linear. Finally, our last assumption was that DT and RF will be prone to overfitting without proper tuning of the hyperparameters.

As of year 2022, most algorithms do not need to be implemented from scratch. In Python, we tend to use the Scikit-learn library[33](Section: 5.1.2) for implementing the desired algorithms. The library contains all necessary implementation documents and guides for how they work and how each parameter affects the output; hence, there was no need to implement the code from scratch. This also made it easier for us to compare other works since the majority of research implements ML algorithms using Scikit-learn.

5.4.1 User-defined Functions (Custom Functions)

When we define functions ourselves to do a specific task, they are referred to as User-defined functions. Functions that already is included with Python are referred to as built-in functions, whereas functions originating from libraries are referred to as library functions. There are several advantages of creating user-defined functions:

1. Maintaining and debugging is easier since the user-defined functions assist in decomposing a big program into smaller sections.
2. Reduce redundancy of code. A function can be called whenever it is needed.
3. Developers who are working on a huge project might divide their work by creating distinct functions.

We have decided to create several user-defined functions in a file named `UNSW_DF.py`. The functions primarily used are for importing datasets. The thesis project has undergone different types of experiments, which require different imports of the dataset, meaning in some cases, we need to import the preprocessed dataset (Listing: 5.9) and, in other cases, the original RAW dataset. We also have created functions where we can plot the Correlation matrix for a given section of a dataset. Since we work with many different files and algorithms, we minimize redundancy by implementing precisely this.

```
def DF_XY():
    """Loads preprocessed dataset, and splits into inputs and output.
    Returns:
        x_train, x_test, y_train, y_test: preprocessed splitted dataset
    """
    try:
        train = pd.read_csv("../Dataset/train_pp3.csv")
        test = pd.read_csv("../Dataset/test_pp3.csv")
        x_train, y_train = train.drop(["label"], axis=1), train["label"]
        x_test, y_test = test.drop(["label"], axis=1), test["label"]
    except:
        print("Could not load dataset, try again..")
    return x_train, x_test, y_train, y_test
```

Listing 5.9: Importing splitted preprocessed dataset

[33]https://scikit-learn.org/stable/

Chapter 6

Performance Evaluation

This Chapter conducts a comprehensive Performance Evaluation for our six selected ML algorithms: ANN, SVM, KNN, LR, DT, and RF. Firstly, we will introduce the performance metrics: Accuracy, Recall, F1 Score, and Precision, which will be utilized to evaluate our experiments. Next, the parameter selection for each algorithm is executed with manual- and Grid Search (Section: 3.4.2). Finally, after gathering the result together and nominating the best-performing model among the six, we extensively compared it with other research, using the same dataset and algorithms. The evaluations for both settings will be presented in tables and visually represented data. Our experiments were conducted on four different computers, where the specifications are presented in Table 6.1.

Table 6.1: Computer Specifications Used for the Thesis

Computer Specifications				
	Processor	**Memory (RAM)**	**Operative System**	**Graphics Card**
1	Intel Core i9-9900K @3.60GHz	32GB DDR4, @3200 MHz	Windows 10 Education	Asus Geforce RTX 2080, 8GB GDDR6
2	Intel Core i7-9700 @3.0 GHz	16GB DDR4 @2667 MHz	Windows 10 Pro	Intel UHD Graphics 630
3	Intel Core i9 9880H @2.3 GHz	32GB DDR4, @2667 MHz	macOS Catalina 10.15.4	AMD Radeon Pro 5500M, 4GB GDDR6
4	Intel Core i7-2600K @ 3.40GHz	16GB DDR3, @1600MHz	Windows 10 Home	Nvidia GeForce GTX 1060, 3GB GDDR5

6.1 Performance Metrics

When evaluating the performance of the models in a Binary Classification problem, we can use the confusion matrix[135], which is shown in this Figure 6.1. This is also called an error matrix which is a two-by-two matrix, and it is used to form the evaluation

metrics[136]. The confusion matrix represents the predicted and actual outcomes and the four outcomes it can have:

1. **True Positive:** True positive is when the predicted outcome is positive, and the actual outcome is positive.
2. **False Positive:** False positive is when the predicted outcome is positive, and the actual outcome is negative.
3. **False Negative:** False negative is when the predicted outcome is negative, and the actual outcome is positive.
4. **True Negative:** True negative is when the predicted outcome is negative, and the actual outcome is negative.

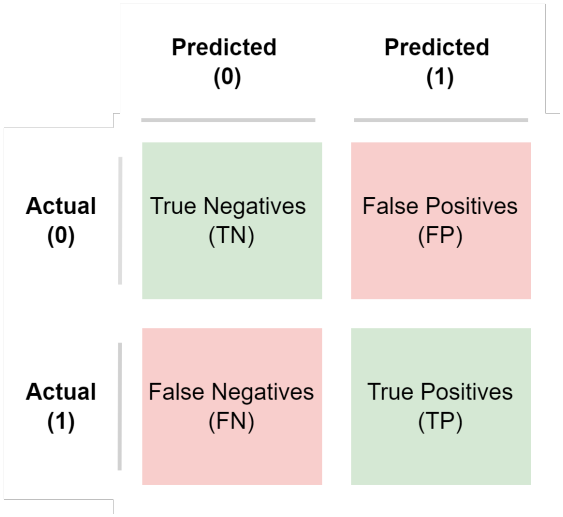


Figure 6.1: Confusion Matrix

The metrics used to evaluate our models are *Accuracy*, *Recall*, *Precision*, *F1* and *Running time*.

Accuracy

Accuracy is utilized to check if the prediction made by the model is correct or not. This is based on all the model's predictions, including the misclassified ones. This metric is calculated by Equation 6.1.

$$Accuracy = \frac{TP + TN}{(TP + TN + FP + FN)} \tag{6.1}$$

Accuracy is best used when the target variables in the data are close to balanced. If data is skewed highly in one direction, such as $90 - 10\%$ (distribution) difference, and the model predicts the same value output for all predictions, the accuracy will be 90%.

Recall

Recall gives us information about what portion of true anomalies were classified. This metric is determined by Equation 6.2.

$$Recall = \frac{TP}{(TP + FN)} \tag{6.2}$$

Recall will check if the classifier can detect whether the outcome is positive or not. If the recall value is close to 100%, it means few false negatives, and if the value is close to 0%, it is few true positives.

Precision

Precision represents the ratio of true positives to all predicted positives. In our case, how many of the output labeled as attacks are actual attacks? This metric is determined by Equation 6.3.

$$Precision = \frac{TP}{(TP + FP)} \tag{6.3}$$

Precision will check if the positive outcome is actually positive or not. If the precision score is low, it tells us that many instances are false positives. A low precision score means few true positives, and a high score means few false positives.

F1 Score

F1 Score uses a mean value method to determine the overall performance of the anomaly detection model by incorporating both Recall and Precision scores. It is a balanced out score for both of the metrics. The reason for having an F1 Score is that recall and precision can affect each other. By reducing one of the metrics, the other can increase and the other way around. This metric is calculated by Equation 6.4.

$$F1\ Score = \frac{2 \times (Recall \times Precision)}{Precision \times Recall} \tag{6.4}$$

In order to get a high F1 Score, both Recall and Precision need to be high, as it goes towards the minimum for them both. The scoring goes between 0 and 100%.

The primary metric used for evaluation in this thesis is accuracy. The datasets' output label is skewed $64 - 36$, and we went for accuracy as our primary metric with F1 Score as a second metric. The difference between training accuracy and the testing accuracy is essential when looking at the accuracy metric. Depending on the importance of the prediction, the difference between training and testing has to reflect the importance. When observing medical predictions such as if a person has a tumor or not, the difference has to be extremely low. However, the difference can be higher when looking at less critical aspects, such as predicting if a person wins a chess match. In our case, we tried to keep the difference between $10 - 11\%$.

6.2 Logistic Regression Results

The evaluation criteria for LR are Accuracy, F1-Score, Precision, and Recall. The metrics are explained in Section 6.1. Firstly, we executed the LR model with default parameters from the Scikit-learn library. The result from the first model was 81.33% for the testing set and 93.54% for the training set. Then we used a manual search to see if it was possible to find hyperparameters that could further increase the performance of LR. The results of the search are represented in Table 6.2, and Figure 6.3. The best results was received with the `newton-cg` solver and a low C value giving an accuracy of 81.83%. Figures 6.2, 6.4 show the distribution for the metrics on the training set.

Table 6.2: Logistic Regression Results

max_Iter	C	Penalty	solver	train_acc	test_acc	F1	precision	recall	runtime(s)
100	1	l2	lbfgs	0.93548	0.81338	0.85257	0.75447	0.97999	2.1
1000	1	l2	lbfgs	0.93522	0.81250	0.85198	0.75351	0.98005	1.2
1000	5	l2	sag	0.93520	0.81206	0.85162	0.75325	0.97955	116
250	3	l1	saga	0.93423	0.80904	0.84927	0.75103	0.97708	44
500	2	l2	lbfgs	0.93587	0.81502	0.85387	0.75560	0.98151	9.2
1500	4	l2	newton-cg	0.93646	0.81585	0.85451	0.75621	0.98218	11.7
100	0.5	none	newton-cg	0.93757	0.81832	0.85639	0.75845	0.98312	36
450	0.3	none	newton-cg	0.93757	0.81832	0.85630	0.75845	0.98312	35
500	0.1	l1	saga	0.93467	0.81020	0.85026	0.75164	0.97866	84
1250	2	l1	saga	0.93500	0.81144	0.85117	0.75270	0.9792	200
1250	2	none	lbfgs	0.93760	0.8184	0.85635	0.75856	0.98308	22
300	0.5	none	lbfgs	0.93693	0.81642	0.85497	0.75658	0.98277	5.7
1600	10	none	lbfgs	0.93759	0.81835	0.85632	0.75849	0.98312	30
1500	10	l2	saga	0.93545	0.81308	0.85243	0.75395	0.98048	181
2000	100	l2	lbfgs	0.93750	0.81798	0.85607	0.75812	0.98305	38

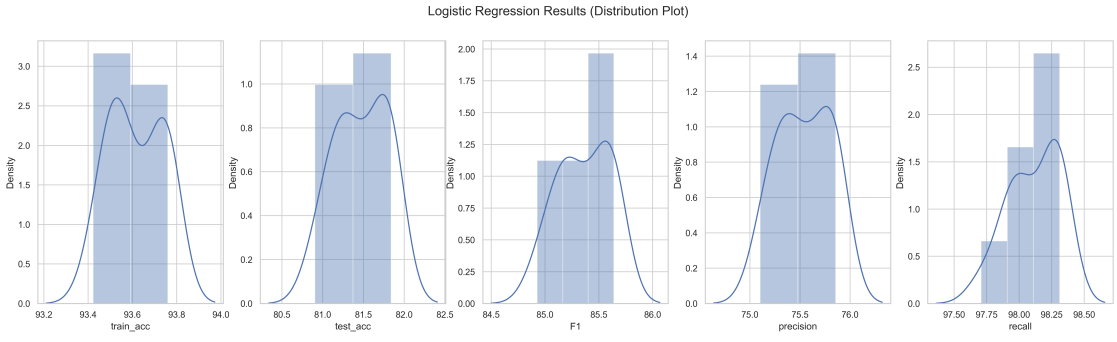


Figure 6.2: Logistic Regression Distribution Plot

The Grid Search algorithm for LR had some limitations due to having matching errors between penalties and solvers. The `sag` and `saga` solver were not included in the grid search because they did not work with several penalties such as `l1` and `elasticnet`. `Sag` and `saga` did not perform well in the manual search, therefore a separate Grid Search

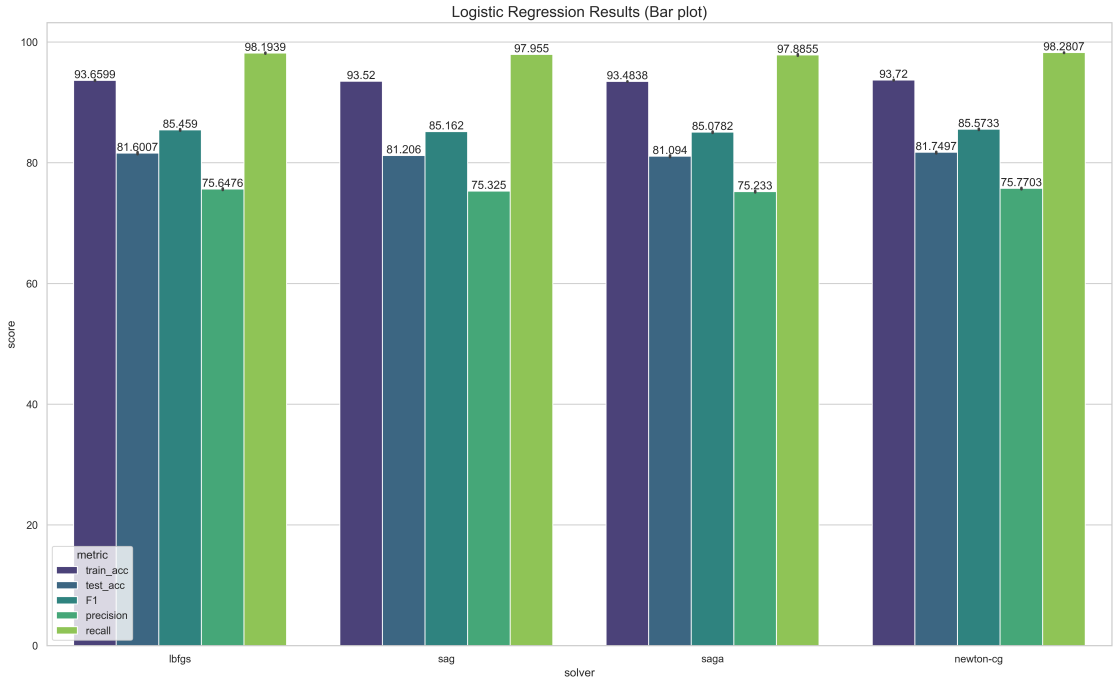


Figure 6.3: Logistic Regression overall Results

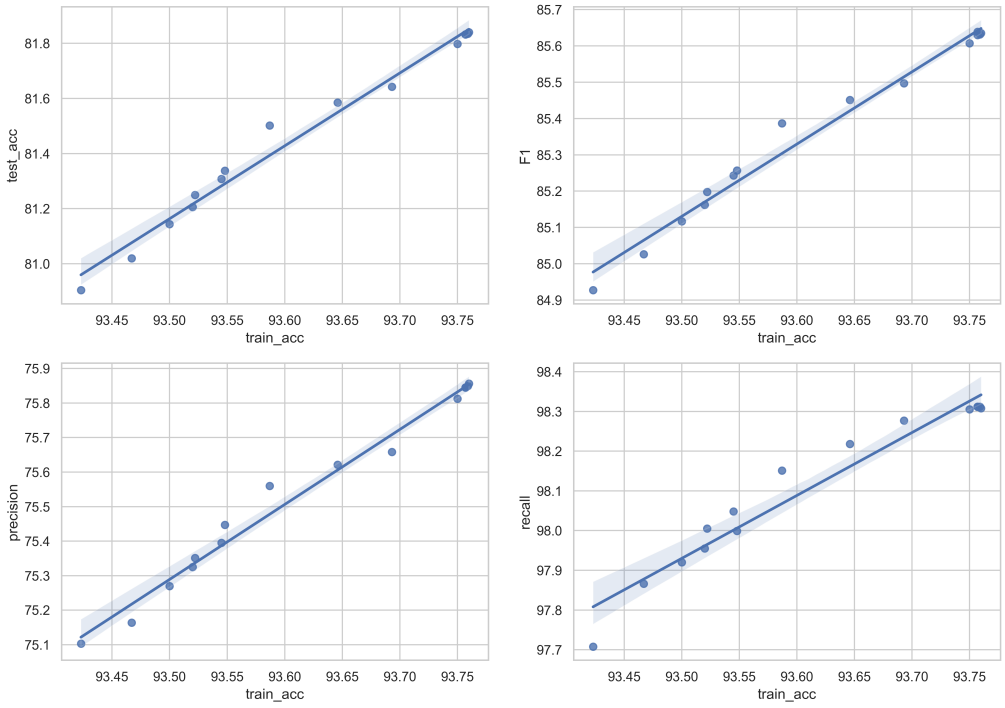


Figure 6.4: Logistic Regression Regression plot

for those solvers was not included. The result of the Grid Search was `solver = liblinear`, `C = 100`, `max_iter = 10`, and `penalty = l1`. With those parameters, the model's accuracy was 81.17%. Where the model's performance did not increase in performance for further Grid Search executions.

Both manual and Grid Search hyperparameter tuning did not highly impact Logistic Regression's performance. The difference between the accuracy with default parameters and the highest accuracy from the manual search is approximately 0.5%. The highest difference between all the models is 0.93%. Meaning any further tuning would most likely not yield considerably more increased results. LR is not the algorithm with the best performance on our NDPF compared to other algorithms, explained later in Section 6.8. However, it did outperform other papers using the same dataset, for example Kasongo et al.[21] which received 79.59%—comparing that against our best result at 81.83%, results in a percentage difference of 2.2%.

6.3 Support Vector Machines Results

The SVM model's performance is affected by multiple different parameters. The parameters can affect the model's run time and accuracy, which depends on the task it will perform. The parameters we tested are different C values, gamma value, and kernels to optimize the model's accuracy on our preprocessed dataset. We used various metrics to evaluate the performance of each model in order to cover the various features of the scoring outcome. The performance metrics used are: Accuracy, Recall, Precision and F1-score, as mentioned in Chapter (6.1).

Table 6.3: Experiments (C 1 To 15 With `rbf` Kernel)

C	Test Accuracy	F1	Precision	Recall	Error
1	0.8157	0.8560	0.7514	0.9945	0.1842
2	0.8194	0.8584	0.7555	0.9937	0.1805
3	0.8220	0.8601	0.7583	0.9933	0.1780
4	0.8234	0.8609	0.7602	0.9924	0.1765
5	0.8249	0.8619	0.7620	0.9920	0.1750
6	0.8258	0.8624	0.7631	0.9914	0.1742
7	0.8266	0.8629	0.7641	0.9910	0.1734
8	0.8273	0.8634	0.7651	0.9906	0.1727
9	0.8279	0.8637	0.7659	0.9902	0.1720
10	0.8288	0.8643	0.7669	0.9900	0.1711
11	0.8294	0.8647	0.7676	0.9899	0.1706
12	0.8297	0.8649	0.7681	0.9896	0.1703
13	0.8303	0.8652	0.7687	0.9895	0.1697
14	0.8304	0.8652	0.7689	0.9892	0.1696
15	0.8305	0.8653	0.7692	0.9889	0.1695

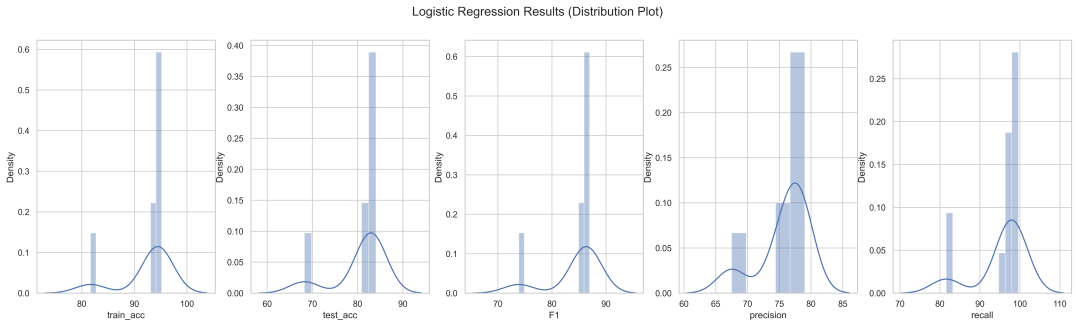


Figure 6.5: Support Vector Machines Distribution Plot

In Figure 6.7, and Table 6.3, we observe that the accuracy of the model increases from 81.57% when C is 1, and up to 83.05% when C is 15. The accuracy steadily rises here. The total run time for the SVM models with C from 1 to 15 was 151 minutes (approximately two and a half hour).

To further experiment with the C value, we executed the model with a C from 1 to 500 to determine how it affected the accuracy. Furthermore, in Figure 6.6, we can observe that the results keep increasing up to 84.04%. As we can see from the Figure, the accuracy declines slightly at specific points. Figure 6.8 represents how the other metrics behaved on C 1 to 500. The reason for the drop in accuracy is explained thoroughly in Chapter 4, Section 4.3. The total run time for C $1 - 500$ was 12765 minutes (approximately 212 hours or 8.8 days).

After experimenting with C values between 1 to 500, we executed manual tuning to increase the model's performance. By testing out multiple kernels, C values, and gamma values in different combinations, we ended up with results shown in Figure 6.9, 6.5, and 6.10. The experiments did not yield any increase in the model's performance. However, most of the results from the different parameters are around 93% on the training set beside the Sigmoid kernel. The last method to get a higher accuracy from the SVM model is to try the Grid Search algorithm. Executing Grid Search on SVM might be problematic because it is time-consuming and computational heavy, even for a single iteration. The Grid Search parameter grid included different kernels (`rbf` and `poly`), different gamma values (multiple values under 1 and `auto` and `scale`), and different C values ($10, 50, 100$ up to 500). The parameters the Grid Search recommended after running resulted in: `C=500`, `gamma = 0.0001`, and the `rbf` kernel, and the accuracy of this model was 93.56% on the training set. Hence, the Grid Search algorithm performed more insufficiently than the manual tuning.

Running experiments on SVM with our preprocessed dataset gave satisfactory results when compared to other projects, for example Kasongo et al. documented an accuracy of 62.42% on its SVM Binary Classification [21]. As a conclusion, our SVM model's highest performance was 84.04% with the expected kernel `rbf`.

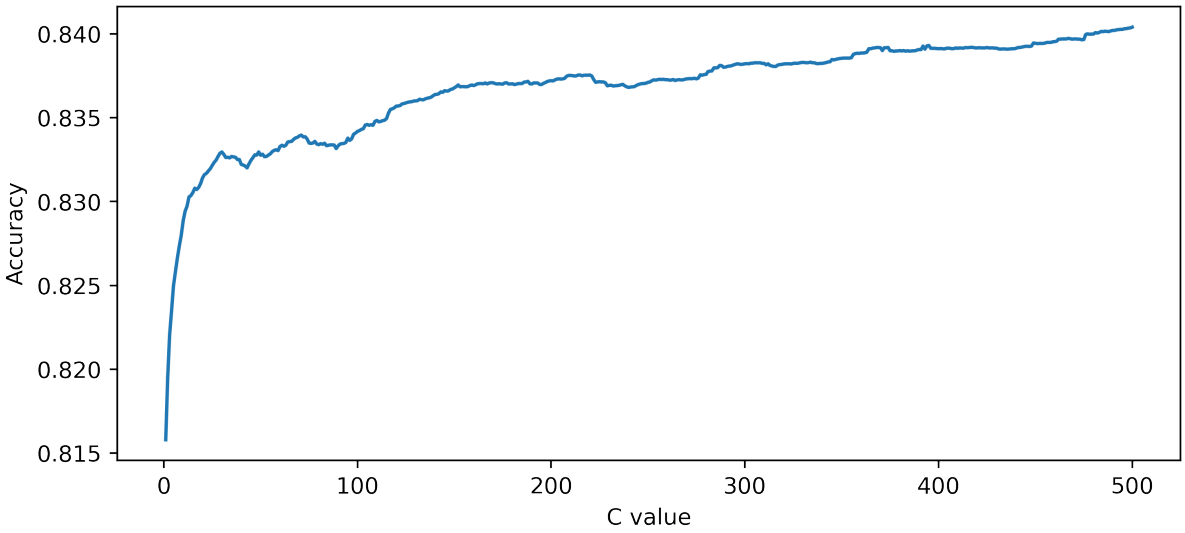


Figure 6.6: SVM With RBF Kernel With a C Value from 1 To 500

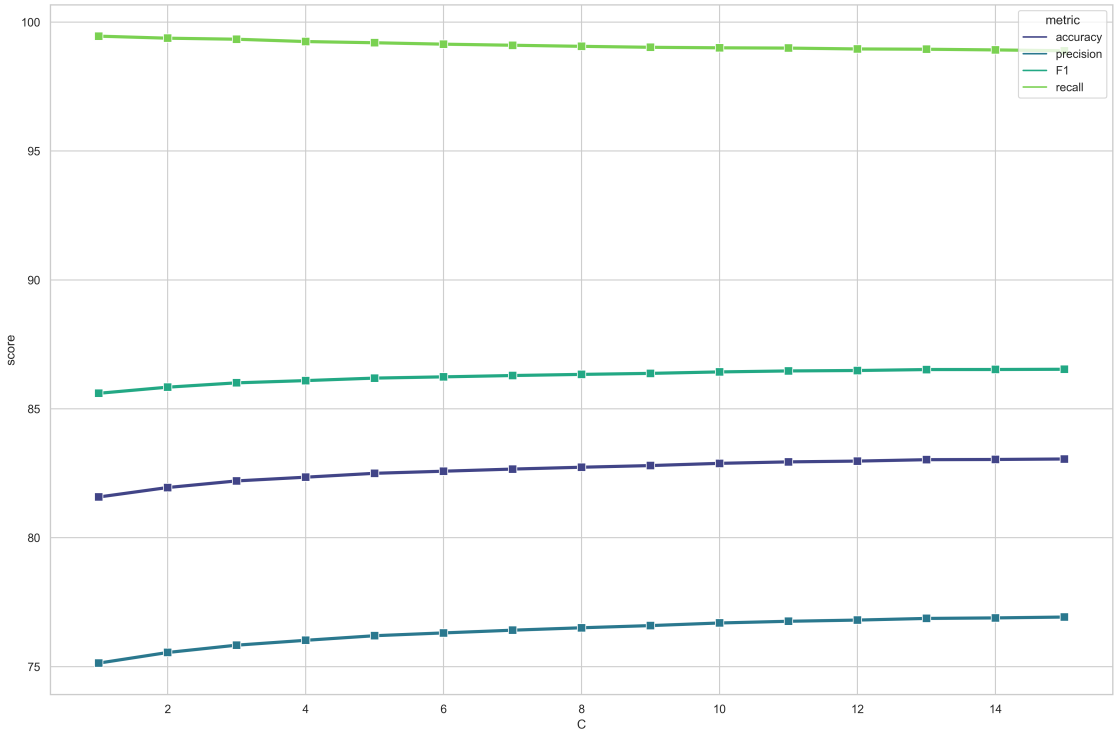


Figure 6.7: Graph Showing How C Value Impacts The Metrics From 1-15

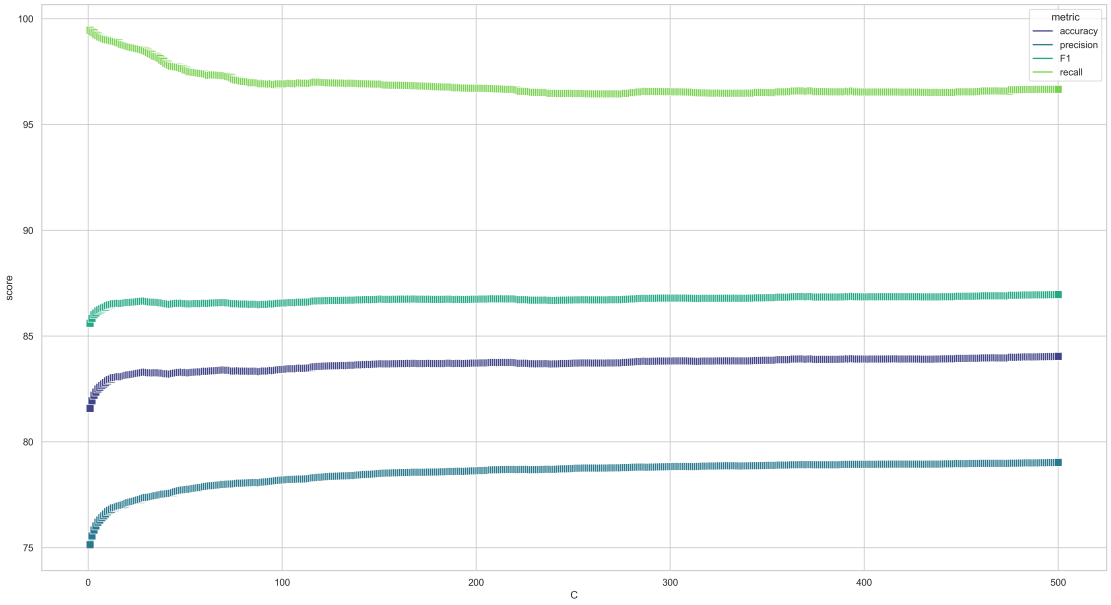


Figure 6.8: Graph Showing How C Value Impacts The Metrics From 1-500

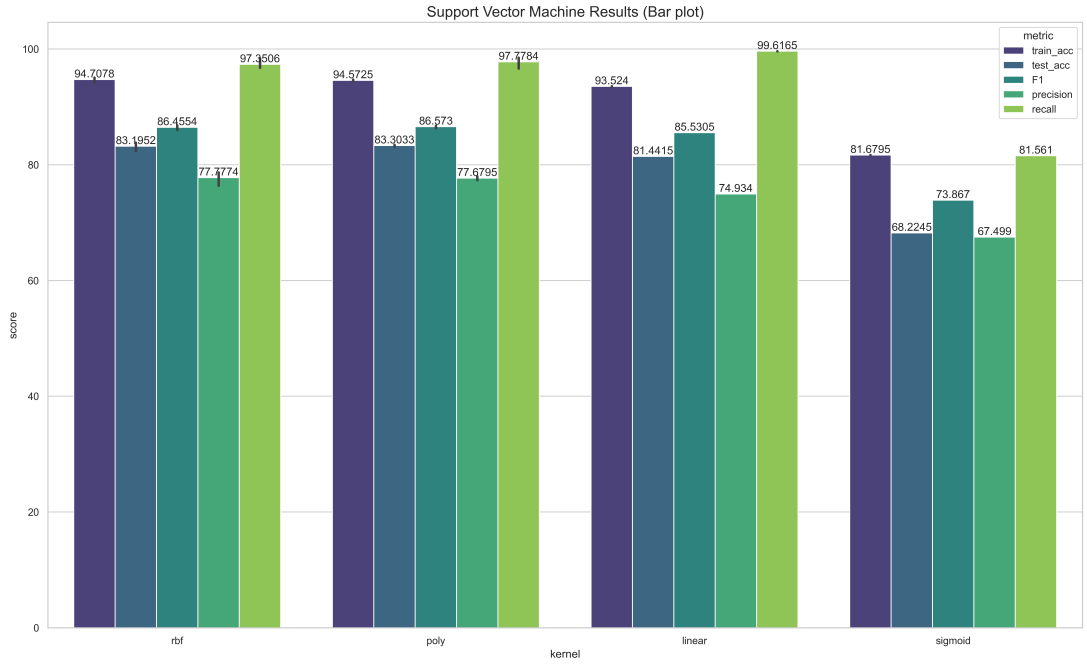


Figure 6.9: Support Vector Machines Overall Results

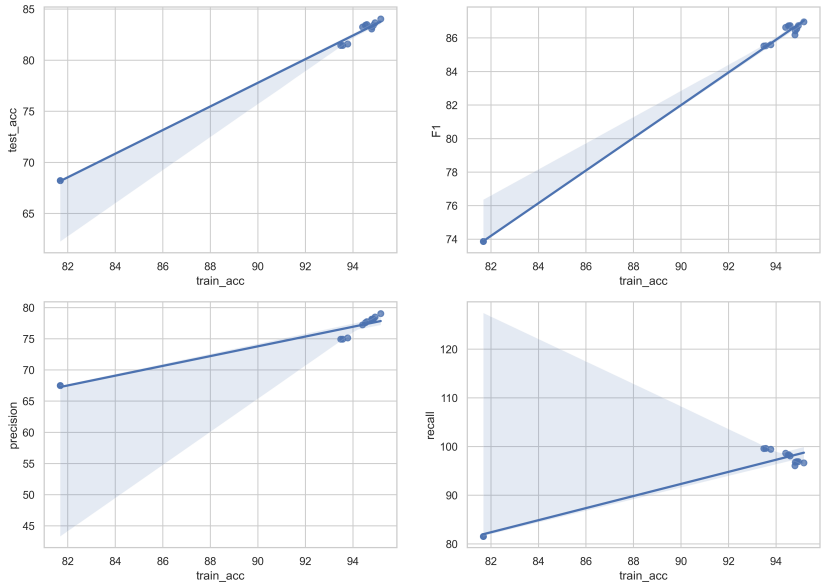


Figure 6.10: Support Vector Machines Regression plot

6.4 Decision Tree Results

When evaluating the DT's performance, we utilize the metrics mentioned in Section 6.1: Accuracy, F1-Score, Precision, and Recall. It is essential to tune DT as it overfits rather quickly. When we executed the DT model with default parameters[1] on our preprocessed dataset, we received an accuracy of 99.81%. The reason for this high accuracy is overfitting which DT is known for when the trees are not pruned. The first method we used to avoid overfitting and increase accuracy was a pre-pruning method for finding an alpha score. After running the cost complexity pruning on the model, it received an alpha value of 0.0002. Adding the alpha value to the model stopped it from overfitting and resulted in an accuracy of 96.95% on the training set, and 86.68% on the testing set. The next step was to execute a manual search with the rest of the parameters and several alpha score values to see if the result could be improved. The accuracy was increased by running the manual search with multiple different values on every parameter. The results and plots from the search are represented in Table 6.4, and Figures 6.12, 6.11, 6.13. The best results were 87.266% on the testing set and 96.41% on the training set.

[1] `https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`

Table 6.4: Decision Tree Results

criterion	min SL	min SS	max depth	max features	ccp_alpha	train_acc	test_acc	F1	precision	recall	runtime(s)
entropy	15	15	50	20	0.000000002	0.96417	0.87257	0.89365	0.82674	0.97233	0.8
entropy	15	15	75	20	0	0.96521	0.86949	0.8917	0.8209	0.97586	0.8
entropy	5	5	90	20	0.002	0.93407	0.82761	0.86379	0.76445	0.9928	1.1
entropy	10	15	20	25	0.000002	0.96275	0.86825	0.89055	0.82065	0.97346	1.8
entropy	20	25	50	30	0.0000002	0.96142	0.87126	0.89294	0.82358	0.97505	1.5
entropy	20	25	55	45	0.0000002	0.96397	0.87266	0.89414	0.82439	0.97679	1.7
entropy	30	30	65	20	0.02	0.91949	0.76377	0.82264	0.70118	0.99499	0.9
gini	1	2	none	none	0	0.99816	0.86406	0.88571	0.82453	0.95669	0.2
gini	5	5	40	5	0.02	0.90438	0.77174	0.82665	0.71035	0.98848	0.4
gini	15	10	45	12	0.002	0.93363	0.80985	0.85266	0.74356	0.99927	0.6
gini	23	17	60	27	0.00002	0.95847	0.86858	0.89103	0.81976	0.97586	1.1
gini	25	25	50	35	0.0000002	0.96174	0.86975	0.8919	0.82122	0.97581	1.2
gini	30	30	50	30	0.000000002	0.95868	0.86997	0.89209	0.82135	0.97617	1.1
gini	45	35	65	25	0.000000002	0.95575	0.86749	0.89131	0.81778	0.97937	0.8

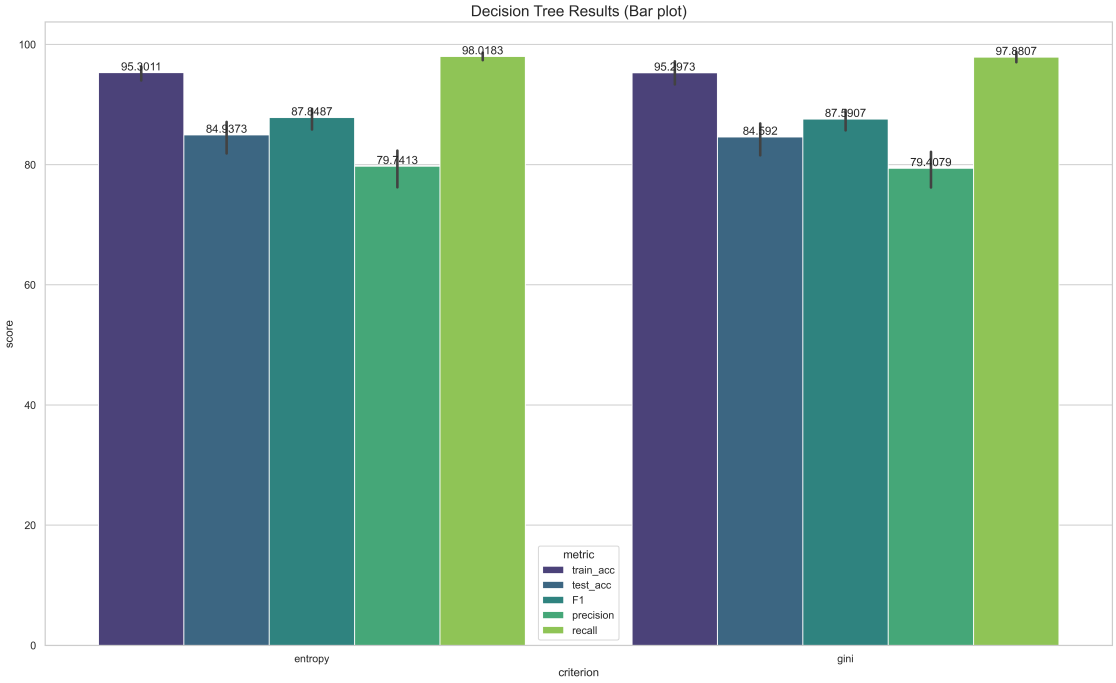


Figure 6.11: Decision Tree overall Results

The last method to try to increase the performance of the DT model is Grid Search. which is explained in Section 3.4. The parameters in Grid Search included: `criterion`, `min_samples_leaf`, `min_samples_split`, `splitter`, and `max_depth`. Due to high memory usage, and time-complexity, we removed several parameters. The remaining parameters consist of $5 - 8$ distinct values. The results from the Grid Search was `criterion = entropy`, `min samples leaf = 17`, `min samples split = 7`, `max depth = 53`, and `splitter = random`. The performance was surprisingly below what it was with default parameters with a testing

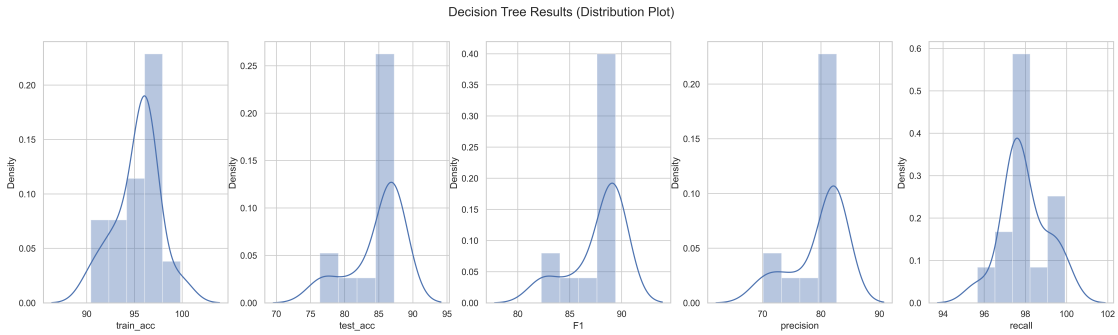


Figure 6.12: Decision Tree Distribution Plot

accuracy of 86.29%, and training accuracy of 95.66%t. This is because the combination of parameters available to the search does not have a combination that yields a higher result. The search did not involve the default parameters since they have already been tested, and executing a search with more parameters would be time-consuming for our thesis, which we could not afford.

The best result from the DT model came from a manual search at 87.266% for the testing set. This performance is comparable to other studies using the same dataset, for example Kasongo et al.[21], which received an accuracy of 88.13%—having less than one percent difference. With a more extensive Grid Search, it should be possible to increase the performance even further to match the results.

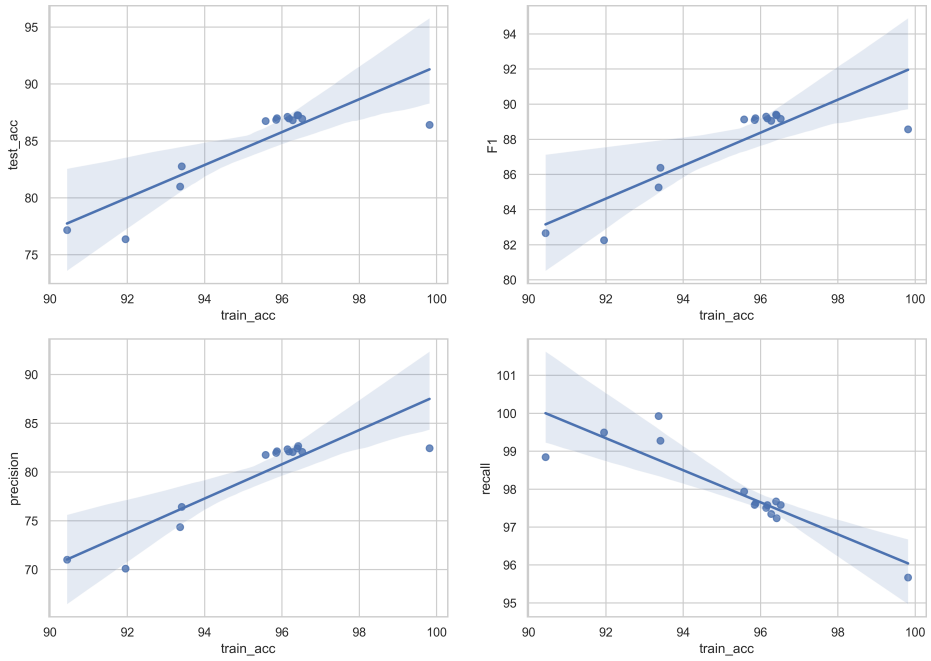


Figure 6.13: Decision Tree Regression plot

6.5 Random Forest Results

The evaluation metrics used for RF is Recall, F1-Score, Accuracy, and Recall, as mentioned in Section 6.1. The first RF algorithm was executed with default hyperparameters[2]. The results of the RF with default parameter is 87.13% on the testing set and 99.81% on the training set. The high difference between the testing and training accuracy, where the training accuracy almost reaches 100%, indicates overfitting. We executed a series of manual searches with different hyperparameter values to resolve the issue of overfitting. The results are represented in Table 6.5, and Figure 6.15. The majority of the final results are partially equal with default parameters, where the training is prone to overfitting. The performance metrics are visually represented on the training set,in Figure 6.14, and 6.16

Table 6.5: Random Forest Results

criterion	n_estimators	max depth	min SS	min SL	train_acc	test_acc	F1	precision	recall	runtime(s)
gini	100	None	2	1	0.99816	0.8713	0.89398	0.81801	0.98551	8.5
gini	200	None	2	1	0.99817	0.87222	0.89473	0.81878	0.98621	16.1
gini	200	20	2	1	0.98625	0.87034	0.89359	0.81512	0.98879	15.8
gini	200	50	2	1	0.99817	0.87213	0.89467	0.81858	0.98637	16.3
gini	200	70	20	10	0.97095	0.86806	0.89204	0.81171	0.99003	14.4
gini	200	100	4	2	0.99338	0.8706	0.89374	0.81566	0.98835	16.6
gini	1000	50	2	1	0.99817	0.87236	0.89491	0.81851	0.98703	76.3
gini	1000	None	2	1	0.99817	0.87203	0.89466	0.81816	0.98694	76.7
gini	2000	50	20	10	0.97117	0.86848	0.89237	0.81215	0.99016	153.4
entropy	100	None	2	1	0.99817	0.87263	0.89491	0.81992	0.985	10.8
entropy	200	None	2	1	0.99817	0.87399	0.89603	0.82094	0.98626	19.3
entropy	200	20	2	1	0.98157	0.87094	0.89396	0.81621	0.98809	18.5
entropy	200	50	2	1	0.99817	0.87329	0.8955	0.82018	0.98606	20.4
entropy	200	70	20	10	0.97192	0.87066	0.89387	0.81524	0.9893	16.6
entropy	200	100	4	2	0.99425	0.87221	0.8949	0.8178	0.98804	17.3
entropy	1000	50	2	1	0.99817	0.87329	0.89554	0.82001	0.98639	83.6
entropy	1000	None	2	1	0.99817	0.87282	0.89518	0.81947	0.9863	84.8
entropy	2000	50	20	10	0.97191	0.87058	0.89381	0.81521	0.98919	166.1

We applied Grid Search with the Random Forest Classifier to improve, and stop the model from overfitting. The hyperparameters involved in the Grid were `bootstrap`, `criterion`, `max_depth`, `max_features`, `min_samples_leaf`, `min_samples_split`, and `n_estimators`. Searching for alpha score and executing cost complexity pruning is unnecessary for RF algorithms as it performs bootstrap aggregation (Refer to Section4.5.2, for more information). The result of the Grid Search algorithm was `bootstrap = "True"` , `max_depth= 50`, `max_features = auto`, `min_samples_leaf = 2`, `min_samples_split = 10`, and `n_estimators = 200`. Utilizing those hyperparameters in the model gave RF an accuracy of 87.21% on the testing set and 98.82% on the training set.

After performing the Grid Search, further testing was needed to see if we could make the accuracy of testing and training not deviate too much. We executed several manual

[2] `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`

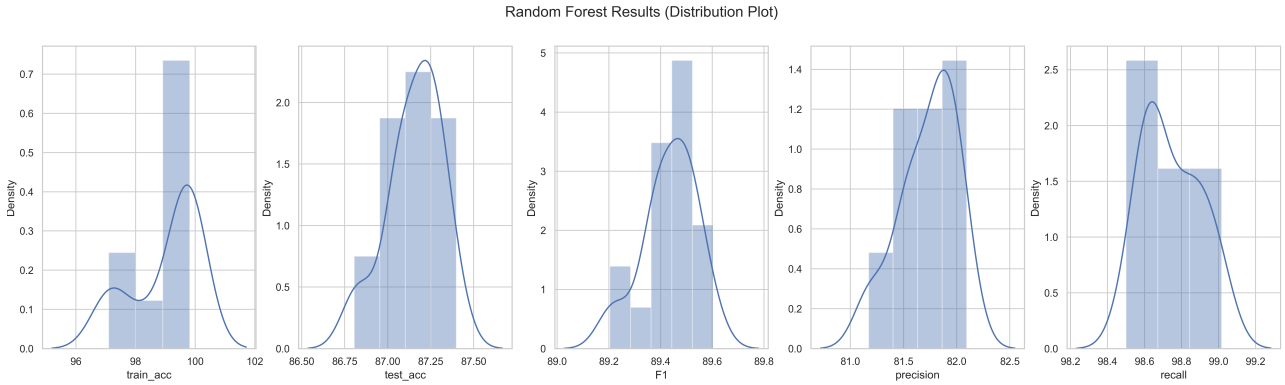


Figure 6.14: Random Forest Distribution Plot

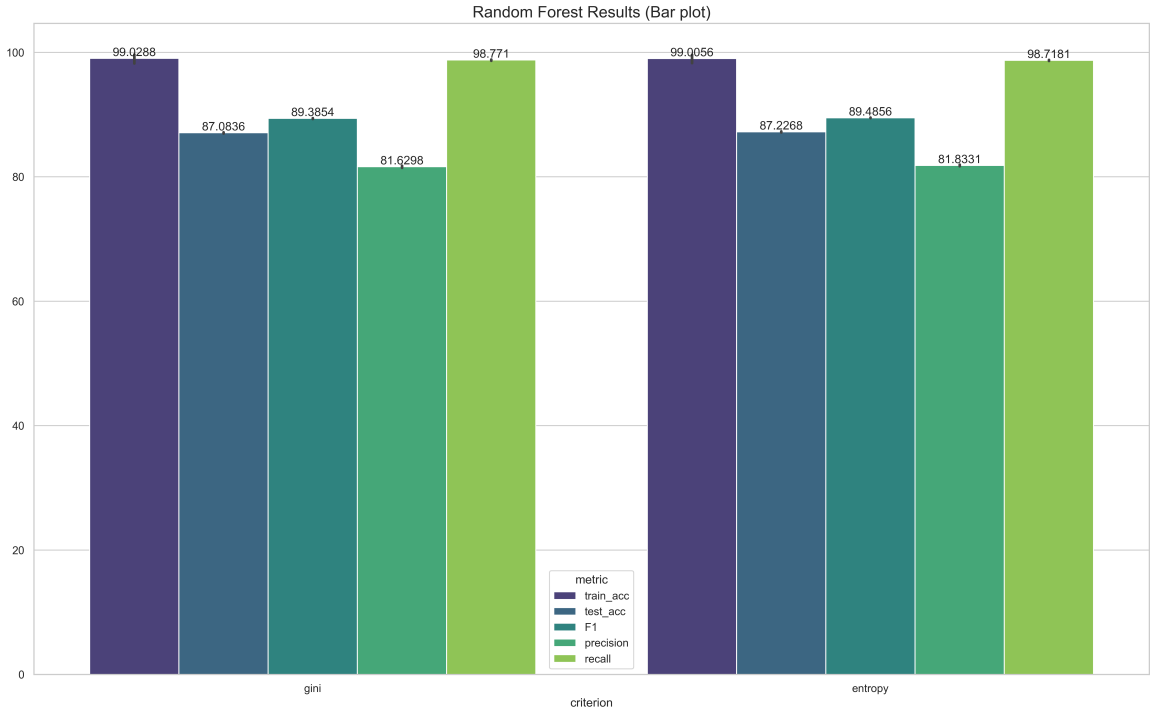


Figure 6.15: Random Forest overall Results

searches with more and deeper trees, which helped with the difference between training and testing accuracy. The results of those experiments were 86.93% on the testing set, and 96.29% on the training set.

The first experiment with RF was run with default parameters and overfitted on the training set. With manual search, the results did not improve much as most training sets still overfit, where the test accuracy did not improve much. The grid search improved the deviation between testing and training accuracy bringing the training accuracy a slightly down and testing accuracy slightly up. Our final experimentations for RF, included more and deeper trees aside from the suggestions from Grid Search algorithm. There we no improvements in the accuracy, but the difference is accuracy between the training and testing datasets was brought down to less than 10%, which is an improvement in itself.

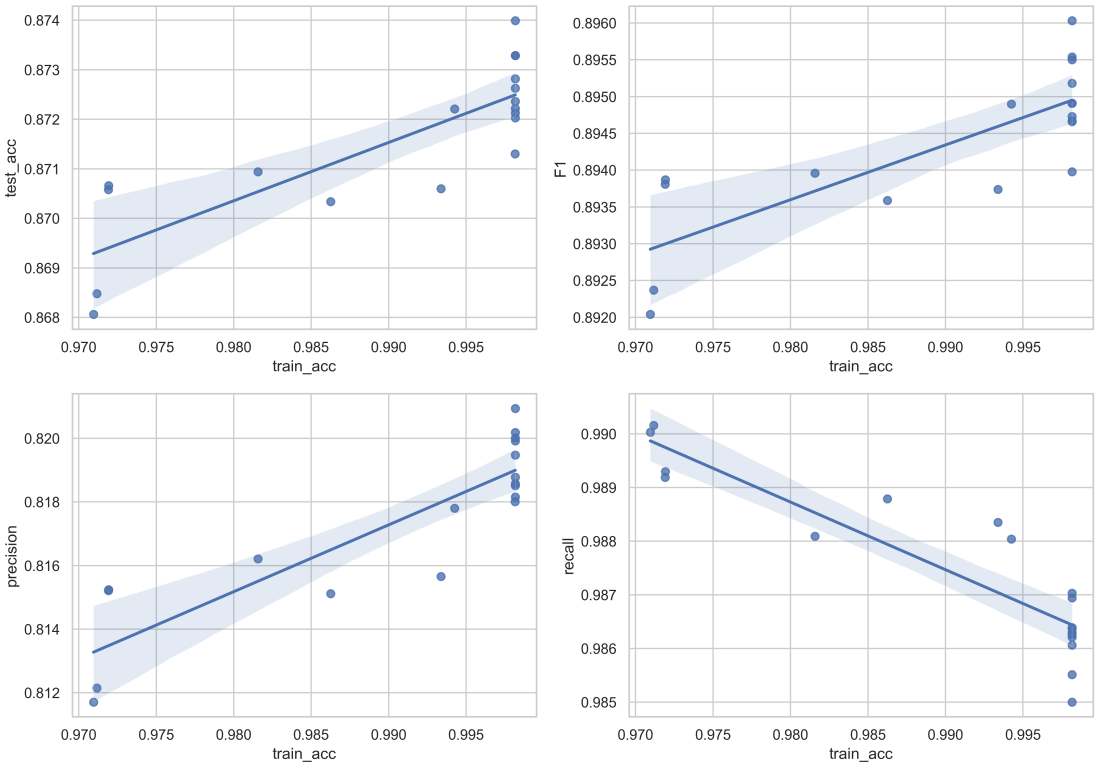


Figure 6.16: Random Forest Regression plot

6.6 Artificial Neural Network Results

A total number of seven experiments were conducted for ANN. Every experiment was run for 200 epochs, with a different number of layers and neurons, which is presented in Table 6.6. Out of the seven experiments, Experiment 7, in Section 6.6.7 received the best overall performance metrics. The model received a training accuracy of 94.796%, a testing accuracy of 88.62%, an F1-score of 96.473%, precision of 84.914%, and recall of 96.473%.

The experiments started with executing the Grid Search algorithm in sections. Firstly, the parameters we tried to output was `epochs` with a range from 80 to 180, `batchsize` with a range from 100 to 200, and `optimizer` with the options `SGD`, `RMSprop`, `Adagrad`, `Adadelta`, `Adam`, `Adamx` and `Nadam`. The Grid Search resulted with the best parameter values of `epochs` = 180, `batchsize` = 100, and `optimizer = adam`. The complete results and scores from Experiment 1 to 7 are presented in Table 6.7 to 6.13 correspondingly. The Distribution plots from Figure 6.17 to 6.23 describe where the range of the selected metrics is heavily distributed.

Table 6.6: The Structure of the Experiments

		Neuron Structure		
Experiment	**Layers**	**Input**	**Hidden**	**Output**
1	3	50	25	1
2	4	53	36-21	1
3	4	500	250-100	1
4	5	100	75-50-25	1
5	7	200	150-125-100-50-25	1
6	9	200	175-150-125-100-75-50-25	1
7	11	300	250-200-175-150-125-100-75-50-25	1

6.6.1 Experiment 1

Table 6.7: Experiment 1: Activation Function →ReLU and Sigmoid

E	L	Train Accuracy	Test Accuracy	Train Error	Test Error	Train Precision	Test Precision	Train F1	Test F1	Train Recall	Test Recall	Runtime(s)
0	3	0.69222	0.56673	0.30778	0.43327	0.69246	0.56048	0.98548	0.98736	0.98548	0.98736	5.867
10	3	0.94535	0.84021	0.05465	0.15979	0.93767	0.77988	0.98519	0.9889	0.98519	0.9889	57.784
20	3	0.94781	0.86173	0.05219	0.13827	0.95201	0.81206	0.97234	0.97439	0.97234	0.97439	103.388
30	3	0.9503	0.87411	0.0497	0.12589	0.95921	0.82863	0.96815	0.97247	0.96815	0.97247	171.73
40	3	0.95061	0.87371	0.04939	0.12629	0.95912	0.82217	0.96872	0.9833	0.96872	0.9833	219.354
50	3	0.95062	0.86631	0.04938	0.13369	0.95766	0.8188	0.97035	0.97238	0.97035	0.97238	270.148
60	3	0.95035	0.87819	0.04965	0.12181	0.96348	0.83472	0.96357	0.97104	0.96357	0.97104	346.828
70	3	0.95273	0.86166	0.04727	0.13834	0.95532	0.80904	0.97621	0.98008	0.97621	0.98008	392.657
80	3	0.95325	0.86835	0.04675	0.13165	0.95864	0.81949	0.97331	0.97584	0.97331	0.97584	419.284
90	3	0.95199	0.87195	0.04801	0.12805	0.96128	0.82777	0.96847	0.96905	0.96847	0.96905	490.22
100	3	0.95313	0.85	0.04687	0.15	0.95299	0.79808	0.97945	0.97399	0.97945	0.97399	625.392
110	3	0.9533	0.88075	0.0467	0.11925	0.96885	0.84109	0.96233	0.96592	0.96233	0.96592	643.316
120	3	0.95359	0.87543	0.04641	0.12457	0.96345	0.82803	0.96856	0.97657	0.96856	0.97657	736.758
130	3	0.95421	0.87237	0.04579	0.12763	0.96407	0.82502	0.96883	0.97498	0.96883	0.97498	743.281
140	3	0.95416	0.86092	0.04584	0.13908	0.95914	0.81165	0.97415	0.97324	0.97415	0.97324	788.155
150	3	0.95502	0.8669	0.04498	0.1331	0.9592	0.81673	0.9754	0.97765	0.9754	0.97765	940.978
160	3	0.95459	0.87214	0.04541	0.12786	0.96282	0.82852	0.97076	0.96817	0.97076	0.96817	959.898
170	3	0.95553	0.87106	0.04447	0.12894	0.96195	0.82399	0.97315	0.97384	0.97315	0.97384	1000.398
180	3	0.9559	0.8654	0.0441	0.1346	0.95981	0.81697	0.97609	0.97368	0.97609	0.97368	1032.018
190	3	0.95409	0.84536	0.04591	0.15464	0.95104	0.78995	0.98316	0.97962	0.98316	0.97962	1074.764
200	3	0.95558	0.8586	0.04442	0.1414	0.95829	0.80727	0.97727	0.97626	0.97727	0.97626	1201.868

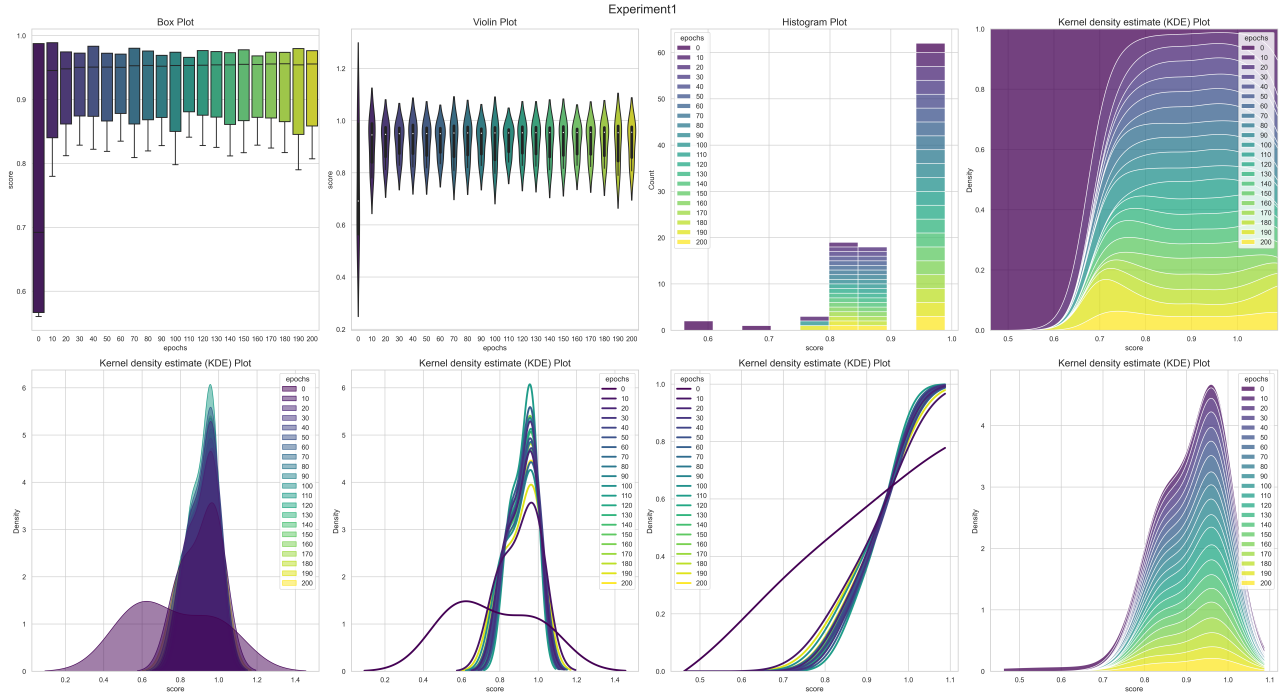


Figure 6.17: Distribution Plot: Experiment 1, 3 Layers

6.6.2 Experiment 2

Table 6.8: Experiment 2: Activation Function →ReLU and Sigmoid

E	L	Train Accuracy	Test Accuracy	Train Error	Test Error	Train Precision	Test Precision	Train F1	Test F1	Train Recall	Test Recall	Runtime(s)
0	4	0.40377	0.41358	0.59623	0.58642	0.61335	0.44619	0.33546	0.2697	0.33546	0.2697	6.442
10	4	0.94596	0.84877	0.05404	0.15123	0.94164	0.79069	0.98142	0.98648	0.98142	0.98648	72.386
20	4	0.94845	0.85765	0.05155	0.14235	0.94852	0.8026	0.9773	0.9833	0.9773	0.9833	134.273
30	4	0.94896	0.85676	0.05104	0.14324	0.94897	0.80203	0.97758	0.98233	0.97758	0.98233	189.392
40	4	0.95163	0.86172	0.04837	0.13828	0.95625	0.81341	0.97347	0.97176	0.97347	0.97176	279.237
50	4	0.95192	0.86919	0.04808	0.13081	0.9615	0.82552	0.96812	0.96676	0.96812	0.96676	316.571
60	4	0.95268	0.86309	0.04732	0.13691	0.95693	0.81343	0.97433	0.97496	0.97433	0.97496	375.444
70	4	0.95468	0.86808	0.04532	0.13192	0.96103	0.82023	0.97286	0.97386	0.97286	0.97386	473.372
80	4	0.95551	0.86591	0.04449	0.13409	0.9627	0.81916	0.97231	0.97077	0.97231	0.97077	502.388
90	4	0.95569	0.85985	0.04431	0.14015	0.95848	0.80899	0.97723	0.97587	0.97723	0.97587	584.255
100	4	0.95553	0.86717	0.04447	0.13283	0.96034	0.81675	0.97492	0.97825	0.97492	0.97825	600.386
110	4	0.95656	0.86755	0.04344	0.13245	0.96357	0.82192	0.97297	0.96949	0.97297	0.96949	660.851
120	4	0.95559	0.86533	0.04441	0.13467	0.9636	0.8221	0.97144	0.96402	0.97144	0.96402	799.548
130	4	0.95784	0.86989	0.04216	0.13011	0.96405	0.82119	0.97439	0.97629	0.97439	0.97629	810.371
140	4	0.95773	0.86461	0.04227	0.13539	0.9623	0.81831	0.97614	0.96932	0.97614	0.96932	904.388
150	4	0.95889	0.86394	0.04111	0.13606	0.96185	0.81253	0.9784	0.97869	0.9784	0.97869	1033.609
160	4	0.95756	0.8654	0.04244	0.1346	0.96437	0.81915	0.97362	0.9696	0.97362	0.9696	986.509
170	4	0.95757	0.85591	0.04243	0.14409	0.95828	0.80659	0.98035	0.97119	0.98035	0.97119	1199.68
180	4	0.95921	0.86143	0.04079	0.13857	0.96659	0.81751	0.97373	0.96338	0.97373	0.96338	1196.063
190	4	0.95902	0.85489	0.04098	0.14511	0.96021	0.80454	0.98042	0.9728	0.98042	0.9728	1247.348
200	4	0.95947	0.86449	0.04053	0.13551	0.96349	0.81835	0.9775	0.96896	0.9775	0.96896	1232.639

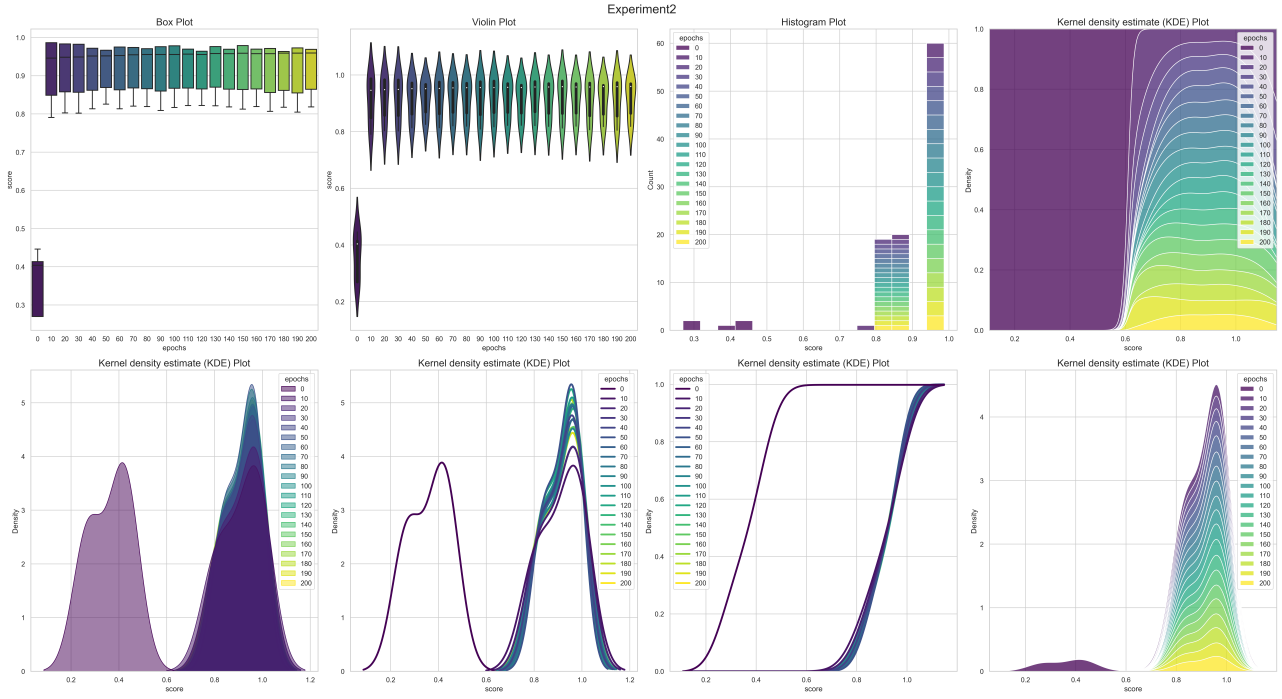


Figure 6.18: Distribution Plot: Experiment 2, 4 Layers

6.6.3 Experiment 3

Table 6.9: Experiment 3: Activation Function →ReLU and Sigmoid

E	L	Train Accuracy	Test Accuracy	Train Error	Test Error	Train Precision	Test Precision	Train F1	Test F1	Train Recall	Test Recall	Runtime(s)
0	4	0.65357	0.53636	0.34643	0.46364	0.80962	0.5867	0.64196	0.53443	0.64196	0.53443	19.652
10	4	0.94664	0.879	0.05336	0.121	0.9638	0.84087	0.95757	0.96237	0.95757	0.96237	143.346
20	4	0.95074	0.85364	0.04926	0.14636	0.94989	0.79891	0.97928	0.98114	0.97928	0.98114	255.482
30	4	0.95475	0.86827	0.04525	0.13173	0.96172	0.82133	0.97221	0.97225	0.97221	0.97225	386.58
40	4	0.95638	0.87556	0.04362	0.12444	0.96851	0.83486	0.96736	0.96486	0.96736	0.96486	542.611
50	4	0.95968	0.86451	0.04032	0.13549	0.96366	0.81666	0.97763	0.97218	0.97763	0.97218	703.872
60	4	0.96183	0.86452	0.03817	0.13548	0.97043	0.82444	0.97359	0.95793	0.97359	0.95793	815.17
70	4	0.96769	0.86325	0.03231	0.13675	0.97205	0.81716	0.98073	0.96828	0.98073	0.96828	928.028
80	4	0.96863	0.85356	0.03137	0.14644	0.96939	0.80767	0.98502	0.96345	0.98502	0.96345	1086.155
90	4	0.9711	0.85229	0.0289	0.14771	0.97124	0.80426	0.98675	0.96711	0.98675	0.96711	1238.601
100	4	0.97282	0.8597	0.02718	0.1403	0.97514	0.8135	0.98519	0.96684	0.98519	0.96684	1409.363
110	4	0.97398	0.85824	0.02602	0.14176	0.97812	0.8137	0.98378	0.96303	0.98378	0.96303	1482.121
120	4	0.97689	0.85422	0.02311	0.14578	0.98063	0.81116	0.9855	0.95835	0.9855	0.95835	1602.806
130	4	0.97808	0.85313	0.02192	0.14687	0.98131	0.80945	0.98658	0.95901	0.98658	0.95901	1685.302
140	4	0.97922	0.85545	0.02078	0.14455	0.98217	0.81101	0.9874	0.96153	0.9874	0.96153	1879.609
150	4	0.98019	0.8562	0.01981	0.1438	0.98233	0.81227	0.98868	0.96093	0.98868	0.96093	1903.517
160	4	0.97893	0.85829	0.02107	0.14171	0.98218	0.81678	0.98695	0.9574	0.98695	0.9574	2166.658
170	4	0.97553	0.86124	0.02447	0.13876	0.98335	0.8234	0.98065	0.95222	0.98065	0.95222	2310.715
180	4	0.98388	0.85008	0.01612	0.14992	0.98522	0.8102	0.99118	0.95034	0.99118	0.95034	2503.043
190	4	0.98581	0.84916	0.01419	0.15084	0.98582	0.80735	0.99344	0.95359	0.99344	0.95359	2603.326
200	4	0.98547	0.86149	0.01453	0.13851	0.98955	0.82149	0.98909	0.95621	0.98909	0.95621	2799.418

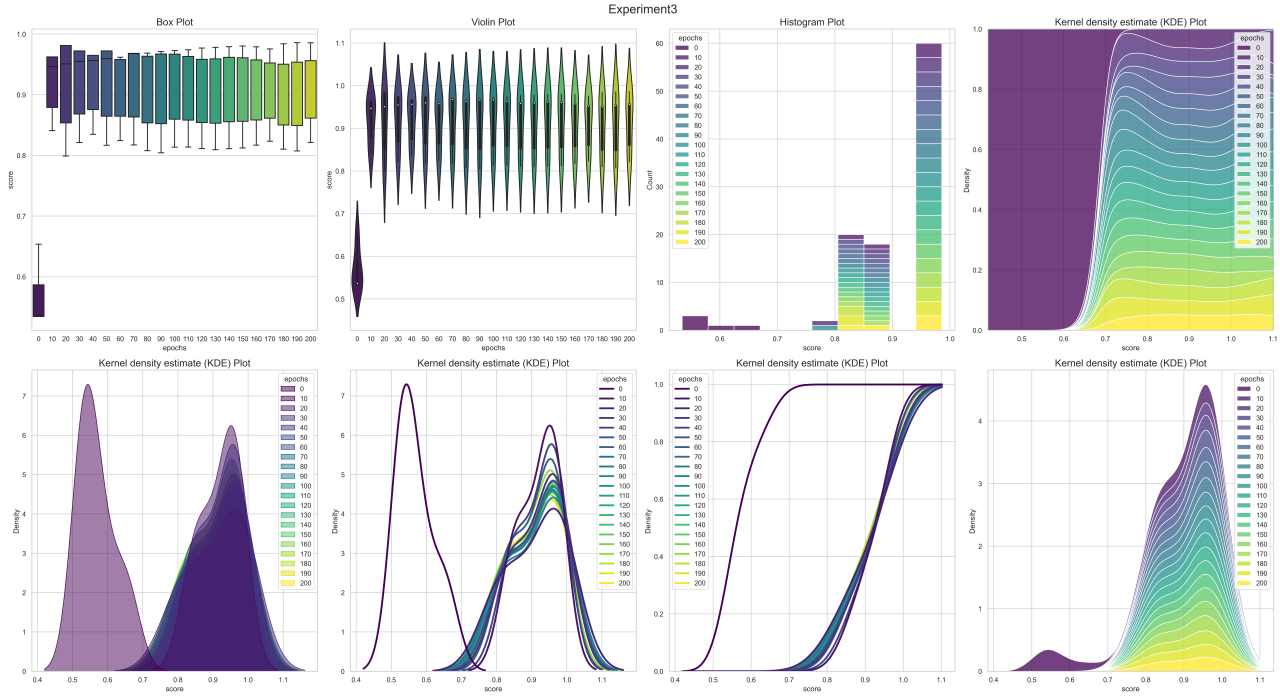


Figure 6.19: Distribution Plot: Experiment 3, 4 Layers

6.6.4 Experiment 4

Table 6.10: Experiment 4: Activation Function →ReLU and Sigmoid

E	L	Train Accuracy	Test Accuracy	Train Error	Test Error	Train Precision	Test Precision	Train F1	Test F1	Train Recall	Test Recall	Runtime(s)
0	5	0.56236	0.52325	0.43764	0.47675	0.74164	0.565	0.54784	0.5829	0.54784	0.5829	14.794
10	5	0.94508	0.88636	0.05492	0.11364	0.96597	0.85178	0.95287	0.9608	0.95287	0.9608	99.899
20	5	0.95019	0.86722	0.04981	0.13278	0.95562	0.81755	0.97195	0.97684	0.97195	0.97684	172.029
30	5	0.95254	0.86486	0.04746	0.13514	0.95821	0.81611	0.97268	0.97404	0.97268	0.97404	257.248
40	5	0.95253	0.86741	0.04747	0.13259	0.95938	0.81959	0.97138	0.97348	0.97138	0.97348	318.451
50	5	0.95759	0.86138	0.04241	0.13862	0.95858	0.81051	0.98003	0.97655	0.98003	0.97655	457.806
60	5	0.95706	0.85979	0.04294	0.14021	0.95886	0.8094	0.97892	0.97492	0.97892	0.97492	556.663
70	5	0.96012	0.86559	0.03988	0.13441	0.9663	0.8192	0.97542	0.96995	0.97542	0.96995	663.043
80	5	0.96042	0.86523	0.03958	0.13477	0.96755	0.81888	0.97454	0.96971	0.97454	0.96971	727.776
90	5	0.9609	0.85874	0.0391	0.14126	0.96552	0.81247	0.97746	0.96654	0.97746	0.96654	776.263
100	5	0.96284	0.86029	0.03716	0.13971	0.96747	0.81425	0.9783	0.9668	0.9783	0.9668	814.886
110	5	0.96453	0.8635	0.03547	0.1365	0.9701	0.81823	0.97804	0.96689	0.97804	0.96689	941.734
120	5	0.9648	0.86163	0.0352	0.13837	0.97078	0.81773	0.97771	0.96345	0.97771	0.96345	1024.809
130	5	0.96772	0.86007	0.03228	0.13993	0.9714	0.81341	0.98147	0.96788	0.98147	0.96788	1197.564
140	5	0.96669	0.86616	0.03331	0.13384	0.97493	0.82469	0.97616	0.96126	0.97616	0.96126	1424.26
150	5	0.96768	0.86304	0.03232	0.13696	0.97676	0.82558	0.97573	0.95248	0.97573	0.95248	1453.881
160	5	0.96786	0.86788	0.03214	0.13212	0.97533	0.82451	0.97751	0.96554	0.97751	0.96554	1401.025
170	5	0.96984	0.85585	0.03016	0.14415	0.97125	0.80788	0.98484	0.96852	0.98484	0.96852	1471.855
180	5	0.97043	0.86465	0.02957	0.13535	0.97701	0.8207	0.97961	0.96499	0.97961	0.96499	1651.516
190	5	0.97012	0.85838	0.02988	0.14162	0.97359	0.81399	0.98276	0.96281	0.98276	0.96281	1893.724
200	5	0.97167	0.86236	0.02833	0.13764	0.97753	0.81907	0.98093	0.96268	0.98093	0.96268	1905.795

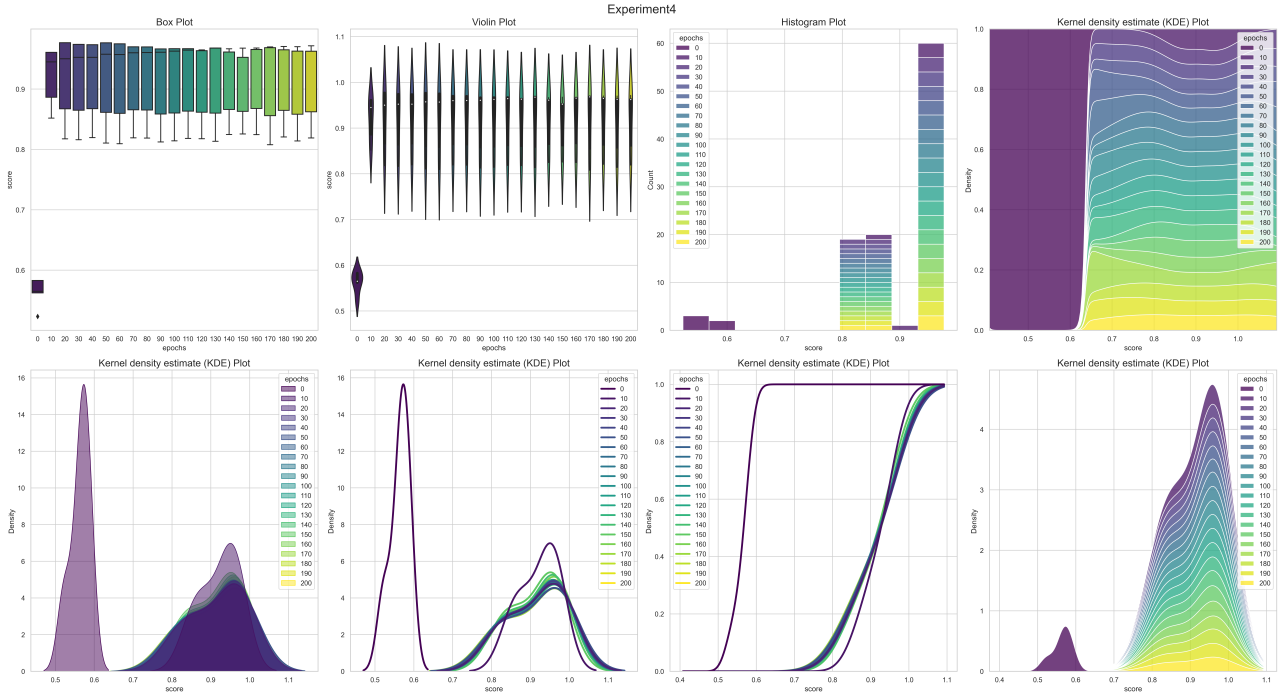


Figure 6.20: Distribution Plot: Experiment 4, 5 Layers

6.6.5 Experiment 5

Table 6.11: Experiment 5: Activation Function →ReLU and Sigmoid

E	L	Train Accuracy	Test Accuracy	Train Error	Test Error	Train Precision	Test Precision	Train F1	Test F1	Train Recall	Test Recall	Runtime(s)
0	7	0.29313	0.38654	0.70687	0.61346	0.42479	0.3209	0.10891	0.10227	0.10891	0.10227	22.068
10	7	0.94734	0.84679	0.05266	0.15321	0.94538	0.79016	0.9792	0.98273	0.9792	0.98273	146.489
20	7	0.95115	0.87288	0.04885	0.12712	0.95776	0.82256	0.97106	0.98068	0.97106	0.98068	251.588
30	7	0.95283	0.85535	0.04717	0.14465	0.95356	0.80264	0.97834	0.9777	0.97834	0.9777	382.763
40	7	0.95591	0.86284	0.04409	0.13716	0.96171	0.81532	0.974	0.97077	0.974	0.97077	511.486
50	7	0.95817	0.87141	0.04183	0.12859	0.96787	0.828	0.97077	0.96742	0.97077	0.96742	610.832
60	7	0.96112	0.8652	0.03888	0.1348	0.96783	0.82084	0.97529	0.96603	0.97529	0.96603	692.81
70	7	0.96386	0.85985	0.03614	0.14015	0.96798	0.81383	0.97929	0.96656	0.97929	0.96656	918.052
80	7	0.96498	0.86675	0.03502	0.13325	0.97343	0.82695	0.97516	0.95857	0.97516	0.95857	1071.846
90	7	0.96801	0.86551	0.03199	0.13449	0.97386	0.82068	0.97929	0.96704	0.97929	0.96704	1206.126
100	7	0.96863	0.86848	0.03137	0.13152	0.97671	0.82805	0.97721	0.96062	0.97721	0.96062	1319.535
110	7	0.97074	0.86381	0.02926	0.13619	0.97507	0.81855	0.98212	0.967	0.98212	0.967	1366.005
120	7	0.97272	0.86256	0.02728	0.13744	0.97975	0.82007	0.98017	0.96129	0.98017	0.96129	1350.649
130	7	0.97388	0.85008	0.02612	0.14992	0.97209	0.79976	0.99005	0.97077	0.99005	0.97077	1506.367
140	7	0.97782	0.86303	0.02218	0.13697	0.98135	0.81904	0.98615	0.96429	0.98615	0.96429	1567.633
150	7	0.97828	0.85439	0.02172	0.14561	0.98029	0.80957	0.98795	0.96179	0.98795	0.96179	1727.497
160	7	0.97942	0.85714	0.02058	0.14286	0.98112	0.81219	0.98879	0.96329	0.98879	0.96329	1888.443
170	7	0.98117	0.85684	0.01883	0.14316	0.98388	0.81291	0.98853	0.9612	0.98853	0.9612	2018.96
180	7	0.97892	0.86414	0.02108	0.13586	0.98408	0.8219	0.98497	0.96162	0.98497	0.96162	2302.948
190	7	0.98257	0.8625	0.01743	0.1375	0.98667	0.82048	0.98773	0.9604	0.98773	0.9604	2346.387
200	7	0.98367	0.85651	0.01633	0.14349	0.98486	0.81296	0.99124	0.96034	0.99124	0.96034	2462.489

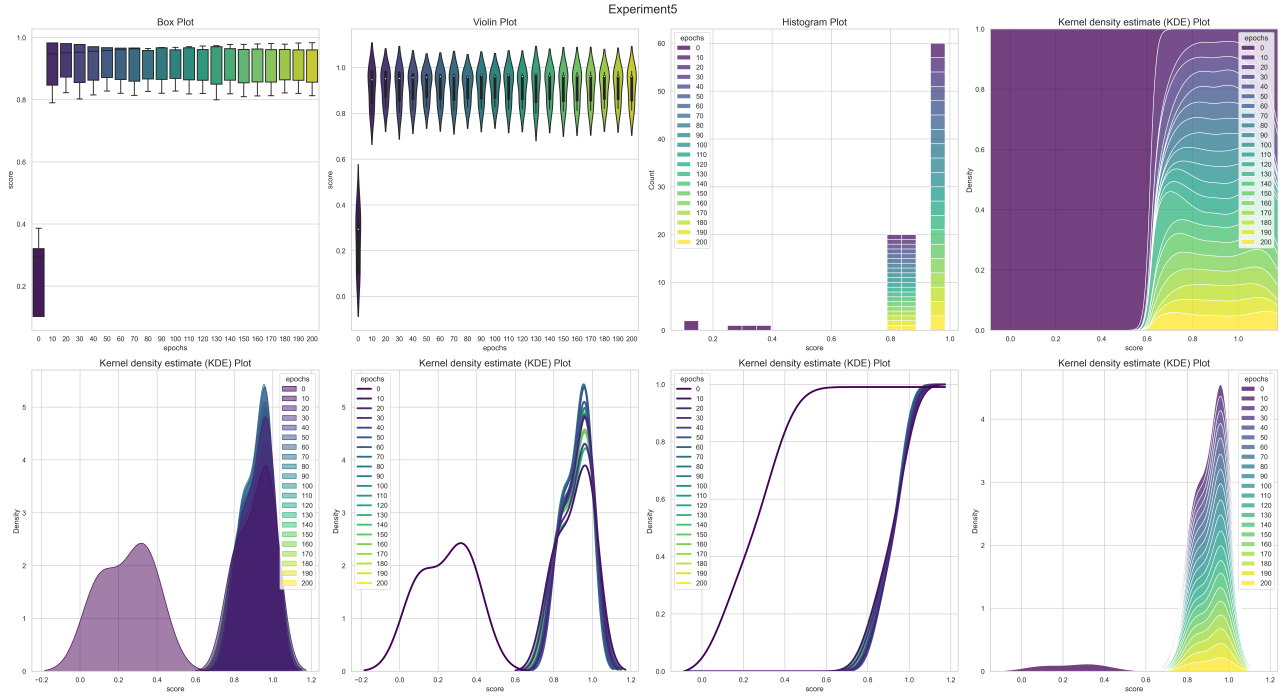


Figure 6.21: Distribution Plot: Experiment 5, 7 Layers

6.6.6 Experiment 6

Table 6.12: Experiment 6: Activation Function →ReLU and Sigmoid

E	L	Train Accuracy	Test Accuracy	Train Error	Test Error	Train Precision	Test Precision	Train F1	Test F1	Train Recall	Test Recall	Runtime(s)
0	9	0.31446	0.44174	0.68554	0.55826	0.10166	0.04993	0.00092	0.00077	0.00092	0.00077	25.631
10	9	0.94733	0.85975	0.05267	0.14025	0.9506	0.8057	0.97319	0.98213	0.97319	0.98213	166.815
20	9	0.95151	0.86587	0.04849	0.13413	0.95665	0.81521	0.97284	0.97812	0.97284	0.97812	298.34
30	9	0.95343	0.8594	0.04657	0.1406	0.95819	0.80997	0.97408	0.97289	0.97408	0.97289	465.589
40	9	0.95415	0.86155	0.04585	0.13845	0.95678	0.8093	0.97676	0.97931	0.97676	0.97931	619.571
50	9	0.95774	0.86346	0.04226	0.13654	0.96459	0.82094	0.97366	0.96179	0.97366	0.96179	746.712
60	9	0.96077	0.86582	0.03923	0.13418	0.96862	0.82244	0.97392	0.96455	0.97392	0.96455	989.225
70	9	0.96466	0.86013	0.03534	0.13987	0.96754	0.81258	0.98098	0.9696	0.98098	0.9696	1094.852
80	9	0.96461	0.86224	0.03539	0.13776	0.96834	0.81414	0.98004	0.97161	0.98004	0.97161	1268.777
90	9	0.96848	0.86298	0.03152	0.13702	0.97426	0.82066	0.97956	0.9612	0.97956	0.9612	1511.303
100	9	0.96945	0.86202	0.03055	0.13798	0.97508	0.81925	0.98017	0.96155	0.98017	0.96155	1608.575
110	9	0.97382	0.86013	0.02618	0.13987	0.9781	0.81521	0.98355	0.96462	0.98355	0.96462	1713.426
120	9	0.97592	0.85556	0.02408	0.14444	0.97667	0.81122	0.98823	0.9614	0.98823	0.9614	1881.962
130	9	0.97326	0.86316	0.02674	0.13684	0.97678	0.8167	0.9841	0.96894	0.9841	0.96894	2108.117
140	9	0.97611	0.8586	0.02389	0.1414	0.97875	0.81291	0.98632	0.96537	0.98632	0.96537	2234.959
150	9	0.97738	0.86443	0.02262	0.13557	0.98175	0.82187	0.98507	0.96234	0.98507	0.96234	2363.852
160	9	0.97636	0.85936	0.02364	0.14064	0.97898	0.81483	0.98645	0.96354	0.98645	0.96354	2413.329
170	9	0.98037	0.86033	0.01963	0.13967	0.98303	0.8162	0.98822	0.96325	0.98822	0.96325	2506.88
180	9	0.97856	0.85801	0.02144	0.14199	0.98302	0.81693	0.98553	0.95645	0.98553	0.95645	2676.926
190	9	0.98383	0.85633	0.01617	0.14367	0.9845	0.81119	0.99185	0.96327	0.99185	0.96327	2746.525
200	9	0.98541	0.86128	0.01459	0.13872	0.98855	0.81946	0.99002	0.95943	0.99002	0.95943	3041.102

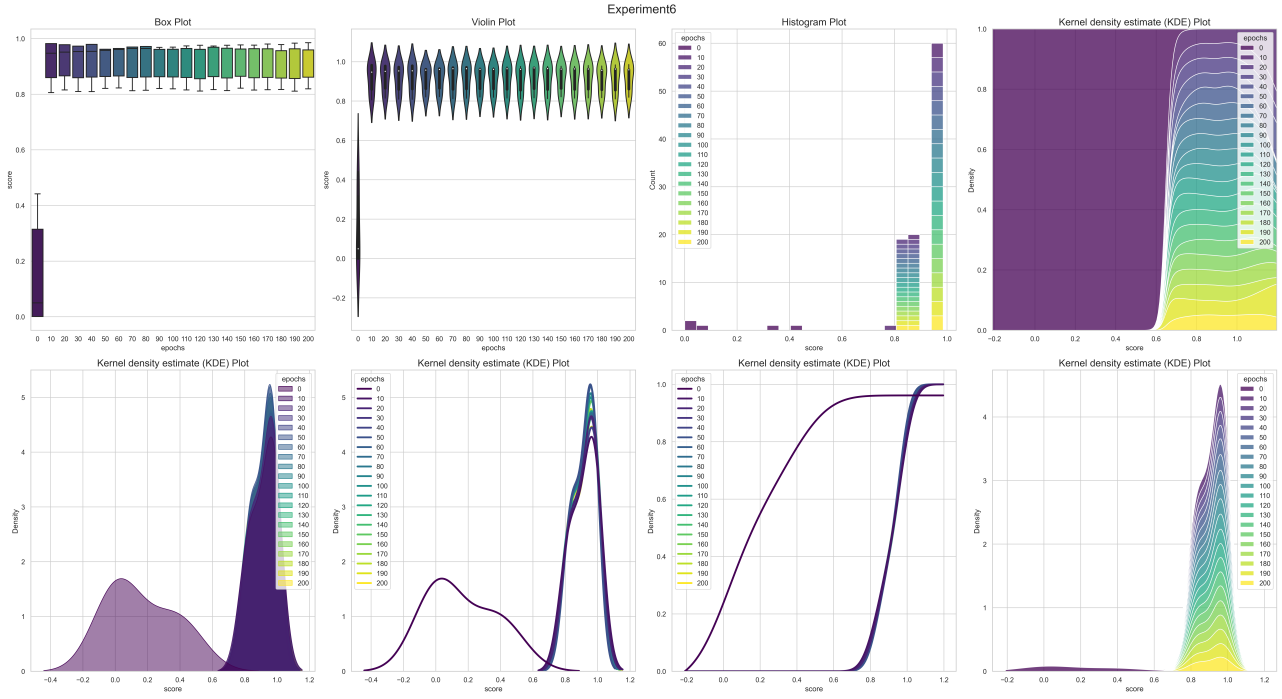


Figure 6.22: Distribution Plot: Experiment 6, 9 Layers

6.6.7 Experiment 7

Table 6.13: Experiment 7: Activation Function →ReLU and Sigmoid

E	L	Train Accuracy	Test Accuracy	Train Error	Test Error	Train Precision	Test Precision	Train F1	Test F1	Train Recall	Test Recall	Runtime(s)
0	11	0.32277	0.44386	0.67723	0.55614	0.59757	0.3007	0.01524	0.00759	0.01524	0.00759	32.085
10	11	0.94622	0.84615	0.05378	0.15385	0.9432	0.78815	0.98001	0.98546	0.98001	0.98546	237.798
20	11	0.94796	0.8862	0.05204	0.1138	0.96806	0.84914	0.95505	0.96473	0.95505	0.96473	455.409
30	11	0.95182	0.87864	0.04818	0.12136	0.96657	0.83673	0.9625	0.96859	0.9625	0.96859	667.354
40	11	0.95432	0.85569	0.04568	0.14431	0.95504	0.80652	0.97897	0.97079	0.97897	0.97079	886.623
50	11	0.95533	0.86375	0.04467	0.13625	0.95911	0.81577	0.97598	0.97207	0.97598	0.97207	1120.929
60	11	0.95979	0.85707	0.04021	0.14293	0.963	0.80922	0.97852	0.96881	0.97852	0.96881	1359.187
70	11	0.96309	0.86342	0.03691	0.13658	0.97026	0.81942	0.97568	0.96448	0.97568	0.96448	1668.132
80	11	0.96177	0.86038	0.03823	0.13962	0.96534	0.81224	0.97898	0.97086	0.97898	0.97086	1796.055
90	11	0.9644	0.86258	0.0356	0.13742	0.97228	0.81782	0.9755	0.9655	0.9755	0.9655	2017.732
100	11	0.97035	0.8641	0.02965	0.1359	0.9758	0.82068	0.98077	0.96376	0.98077	0.96376	2164.269
110	11	0.97137	0.86415	0.02863	0.13585	0.97759	0.82027	0.98041	0.96462	0.98041	0.96462	2486.649
120	11	0.97256	0.8567	0.02744	0.1433	0.9753	0.81167	0.98462	0.96325	0.98462	0.96325	2628.038
130	11	0.96974	0.85569	0.03026	0.14431	0.97206	0.80766	0.98383	0.96857	0.98383	0.96857	2932.004
140	11	0.97816	0.85596	0.02184	0.14404	0.97959	0.81075	0.9885	0.96325	0.9885	0.96325	3294.701
150	11	0.97137	0.86206	0.02863	0.13794	0.97753	0.81868	0.98048	0.96268	0.98048	0.96268	3505.01
160	11	0.97576	0.86599	0.02424	0.13401	0.98274	0.82576	0.98162	0.95897	0.98162	0.95897	3637.188
170	11	0.97672	0.85878	0.02328	0.14122	0.97788	0.81083	0.98814	0.96976	0.98814	0.96976	3898.866
180	11	0.97654	0.8586	0.02346	0.1414	0.98088	0.81307	0.98472	0.96506	0.98472	0.96506	4229.4
190	11	0.97507	0.86475	0.02493	0.13525	0.98271	0.82556	0.98062	0.95648	0.98062	0.95648	4429.623
200	11	0.97366	0.85416	0.02634	0.14584	0.97681	0.8076	0.98468	0.96504	0.98468	0.96504	4379.668

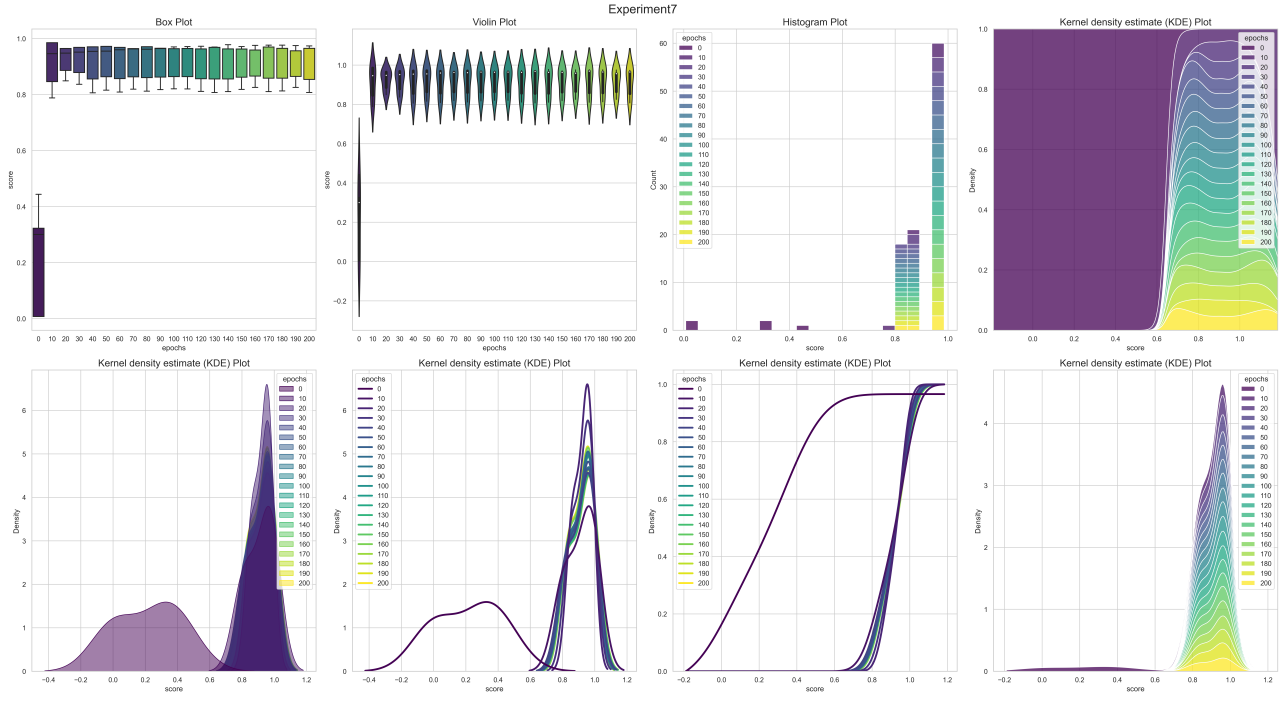


Figure 6.23: Distribution Plot: Experiment 7, 11 Layers

6.7 K-Nearest Neighbor Results

The KNN algorithm was implemented on the preprocessed UNSW-NB15 dataset, where we used the performance metrics, Accuracy, F1-Score, Precision, and Recall as mentioned in Section 6.8 for evaluating the output.

Table 6.14 presents the result of our first experiment with the values of K from 1 to 70. Furthermore, the model became less stable due to majority voting and averaging, resulting in a less accurate algorithm.

For example, the average training accuracy for $K(1-70)$ equals 94.6635%, whereas the testing accuracy averaged 83.5129% for the whole experiment. By analyzing the results from Table 6.14 and Figures 6.24 and 6.25 plotted from the output, we can observe that $K = 1$ is overfitting, because the model assigns every first neighbor it encounters, and uses for prediction. Furthermore, at $K = 2$ in Figure 6.27, we can observe that the error is minimal. We can conclude that the predictions become less stable and accurate as the K value increases for the K range from 1 to 70. To observe closer, we have included Figure 6.28 which represents the distribution of the metrics for K 1 to 15.

Table 6.14: KNN Scores with K from 1 to 70

K	train_acc	test_acc	precision	F1	recall	K	train_acc	test_acc	precision	F1	recall
1	99.71199	85.27911	81.48893	87.64123	94.79838	**36**	94.40576	83.3552	77.53901	86.66355	98.22201
2	96.52163	87.10829	85.70341	88.70274	91.91962	**37**	94.39036	83.04183	77.10104	86.47198	98.43598
3	96.71668	85.08964	80.58249	87.64692	96.069	**38**	94.37838	83.28232	77.44662	86.6161	98.24848
4	95.8498	86.31881	83.19691	88.34485	94.17189	**39**	94.34188	83.01511	77.0596	86.45696	98.46466
5	95.87661	84.62809	79.77909	87.36876	96.55431	**40**	94.34644	83.23859	77.36393	86.59517	98.32789
6	95.43803	85.62284	81.68095	87.94541	95.2506	**41**	94.32648	83.00418	77.0271	86.4552	98.51319
7	95.48081	84.36938	79.27285	87.23073	96.96462	**42**	94.32306	83.18515	77.29448	86.56105	98.35216
8	95.27492	85.25847	80.84063	87.75883	95.97194	**43**	94.31508	82.91187	76.9244	86.39217	98.5176
9	95.2407	84.14468	78.87946	87.10283	97.24036	**44**	94.32477	83.1062	77.18394	86.51301	98.40731
10	95.07645	84.87101	80.11616	87.53328	96.46387	**45**	94.30994	82.88758	76.88032	86.38047	98.55952
11	95.02683	84.04266	78.67092	87.0531	97.43448	**46**	94.29626	83.04183	77.11416	86.46831	98.4051
12	94.98463	84.67303	79.74179	87.42213	96.73961	**47**	94.27002	82.84992	76.82826	86.35775	98.58599
13	94.92874	83.92241	78.45655	86.98747	97.59993	**48**	94.282	83.00661	77.05122	86.45084	98.46245
14	94.87798	84.40825	79.29325	87.2645	97.01756	**49**	94.24778	82.77462	76.75106	86.30473	98.57496
15	94.866	83.75601	78.21251	86.88438	97.71905	**50**	94.27059	82.89608	76.93037	86.37472	98.46245
16	94.84205	84.23213	78.99265	87.16204	97.21609	**51**	94.25405	82.7564	76.71477	86.29699	98.61467
17	94.78958	83.68314	78.07132	86.84849	97.8492	**52**	94.26603	82.90944	76.92268	86.39023	98.5154
18	94.77019	84.14954	78.80367	87.12637	97.41463	**53**	94.24493	82.73818	76.67581	86.29006	98.66099
19	94.71259	83.64184	77.95873	86.83686	97.997	**54**	94.26318	82.8815	76.87278	86.37655	98.56172
20	94.71544	84.05359	78.61624	87.07763	97.58008	**55**	94.22554	82.73332	76.66141	86.28937	98.68305
21	94.68749	83.59447	77.8751	86.81176	98.06538	**56**	94.21641	82.8815	76.85985	86.38024	98.59261
22	94.68806	83.88597	78.37761	86.9713	97.68155	**57**	94.2147	82.71996	76.6351	86.28366	98.71173
23	94.64814	83.50095	77.75078	86.75171	98.1095	**58**	94.2204	82.85843	76.82459	86.36728	98.61687
24	94.65271	83.81553	78.24729	86.93461	97.79185	**59**	94.18562	82.70782	76.61377	86.27772	98.73158
25	94.58142	83.44508	77.6767	86.71592	98.13597	**60**	94.19816	82.81592	76.77014	86.34046	98.63672
26	94.574	83.75844	78.14173	86.90509	97.88229	**61**	94.17307	82.68353	76.57937	86.26348	98.75143
27	94.5107	83.40985	77.60655	86.69971	98.20656	**62**	94.18904	82.77341	76.72587	86.31077	98.63231
28	94.50499	83.67585	78.03052	86.85189	97.922	**63**	94.16851	82.61308	76.5089	86.2137	98.7382
29	94.46792	83.27989	77.44774	86.61338	98.23965	**64**	94.17877	82.75883	76.69542	86.305	98.66761
30	94.49986	83.54346	77.84124	86.76998	98.01244	**65**	94.16223	82.62158	76.50899	86.22217	98.76026
31	94.4742	83.2471	77.37998	86.59924	98.31245	**66**	94.16794	82.72968	76.66478	86.28476	98.6654
32	94.47648	83.50095	77.74884	86.75223	98.11392	**67**	94.13543	82.58393	76.46305	86.19803	98.77349
33	94.42173	83.17301	77.27257	86.555	98.37201	**68**	94.15539	82.70539	76.62847	86.27103	98.68967
34	94.428	83.36977	77.59559	86.66277	98.12936	**69**	94.13315	82.57907	76.45151	86.19658	98.78893
35	94.40291	83.10863	77.19414	86.51259	98.38966	**70**	94.14455	82.69931	76.61644	86.26846	98.7029

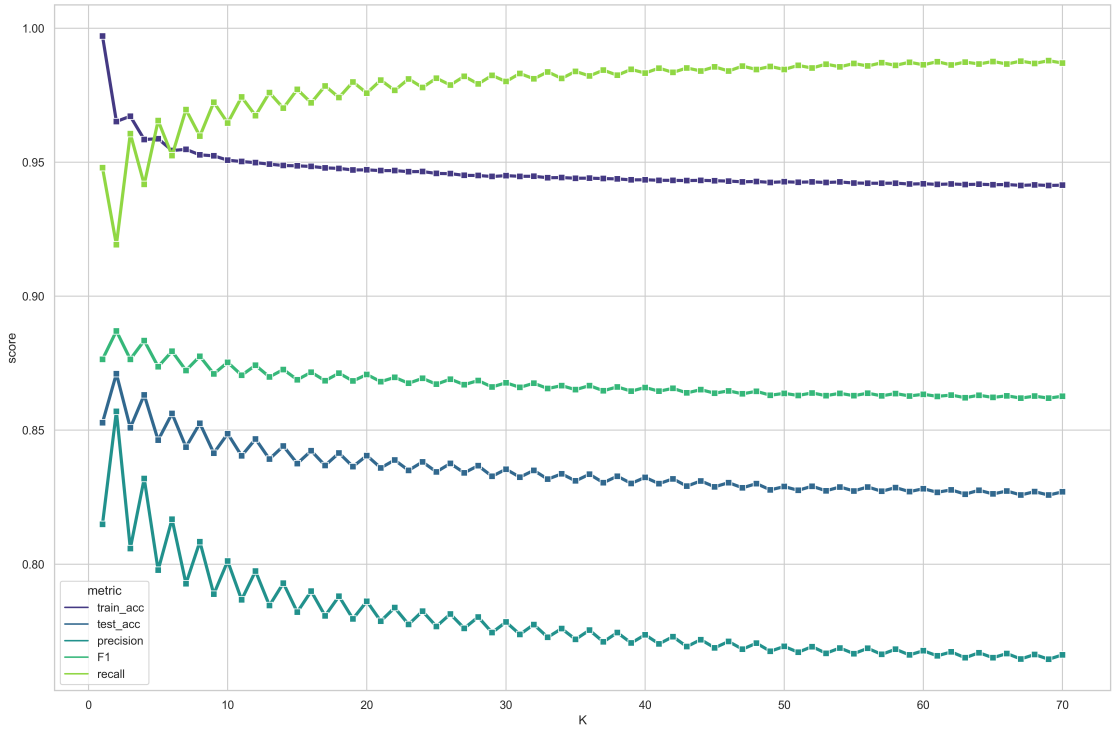


Figure 6.24: Lineplot with respect to K value from 1 to 70

The error rate for the model is also calculated and presented in Figure 6.26. We can analyze that the error rate increases as K increases. The error rate E is calculated with Equation 6.5:

$$\text{Model Error} = (100 - accuracy) \tag{6.5}$$

The decrease in accuracy as the K value increases brought up a question, which refers to the next experiment to confirm our assertion: investigate whether a higher K value than 70 results in better or worse model performance. We conducted the same experiment with K ranging from 1 to 200. We can analyze from Figure 6.27 that our assertion can be confirmed: The KNN Algorithm's performance becomes less stable and accurate as the K-value increases. The total runtime for this experiment was 9672 minutes, approximately 161.2 hours.

Finally, the KNN Algorithm implementation and experiments resulted in the best score at $K(2)$. The model received a training accuracy of 96.52163%, testing accuracy of 87.10829%, Precision of 85.70341%, F1-Score of 88.70274%, and Recall of 91.91962% respectively.

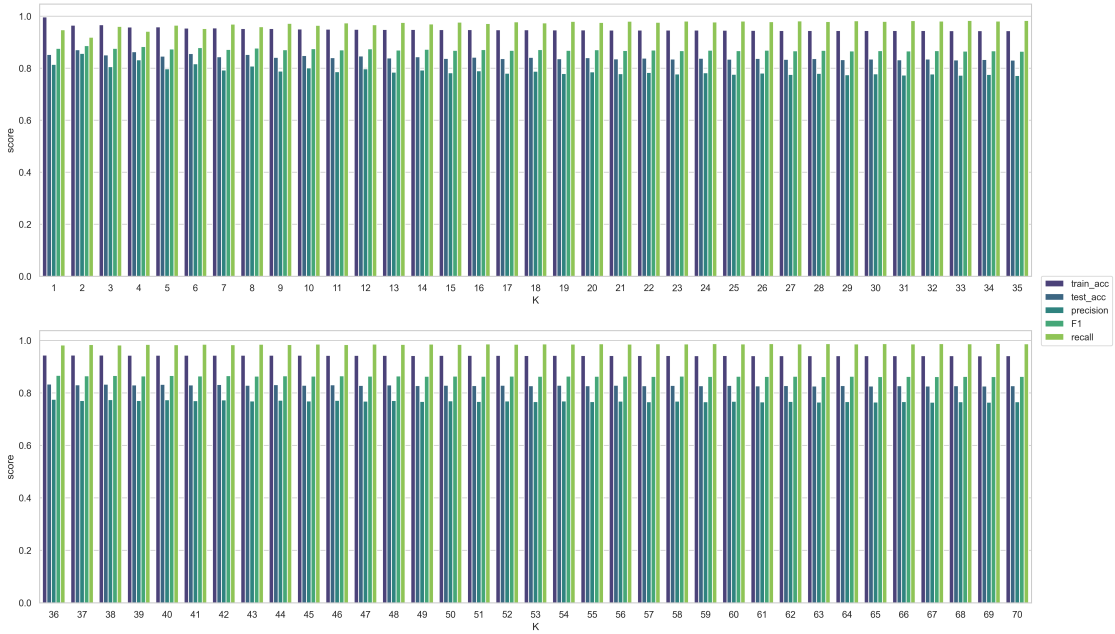


Figure 6.25: Barplot with respect to K value from 1 to 70

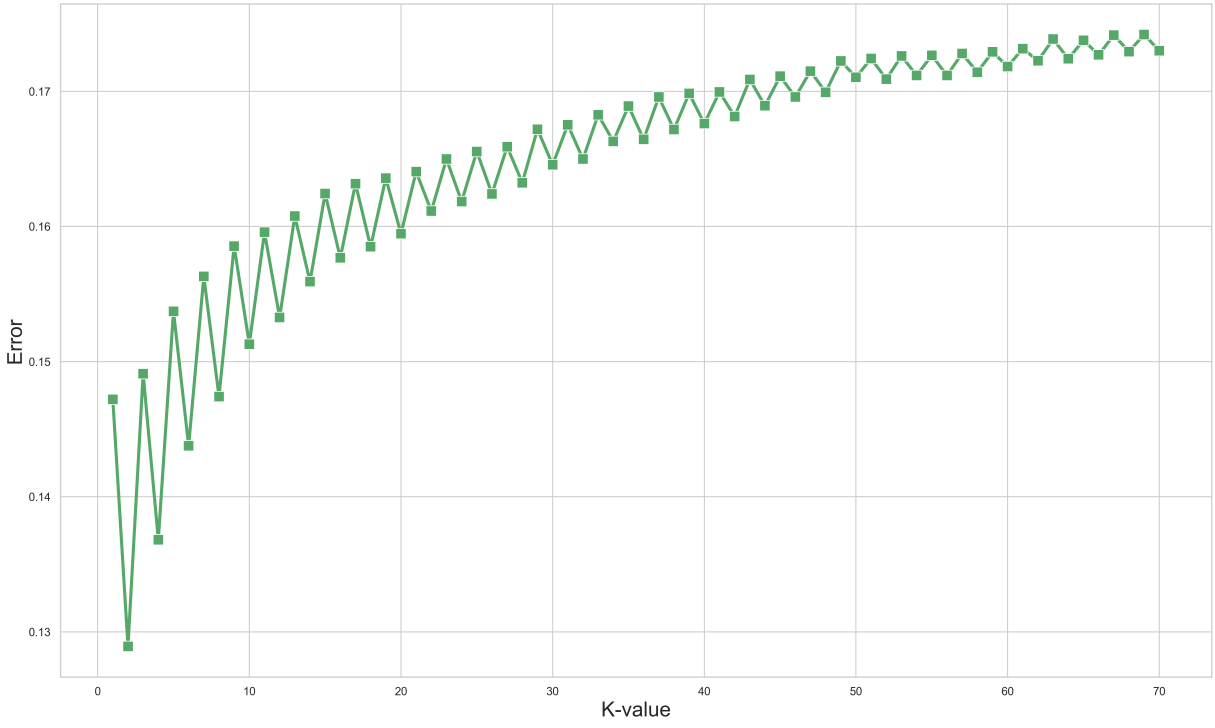


Figure 6.26: Error rate with respect to K value from 1 to 70

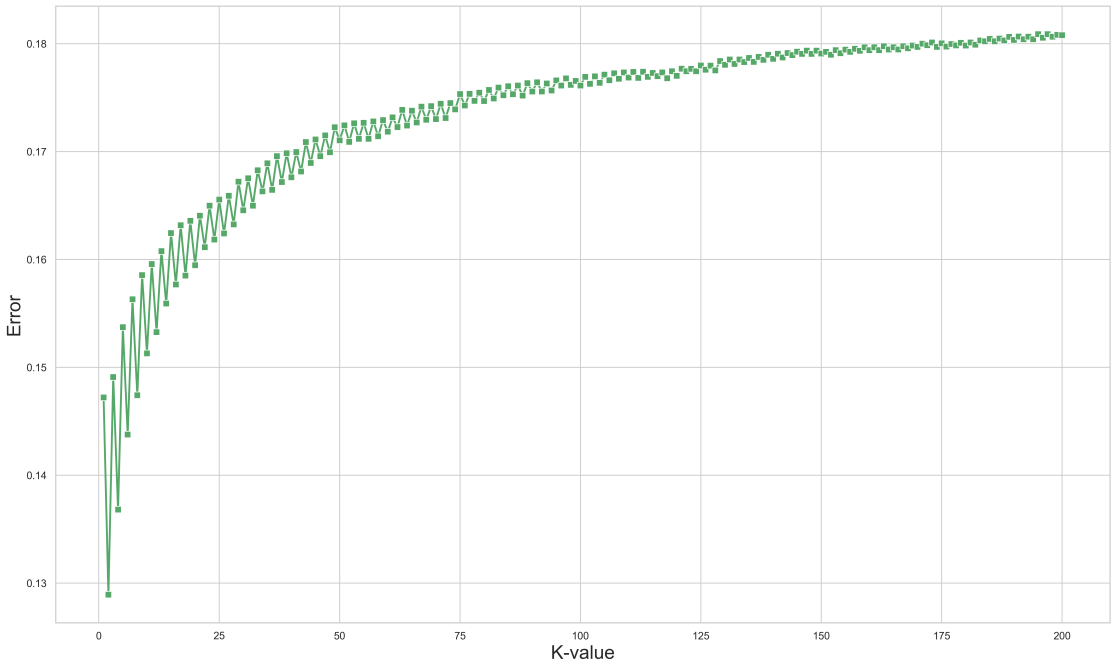


Figure 6.27: Error rate with respect to K value from 1 to 200

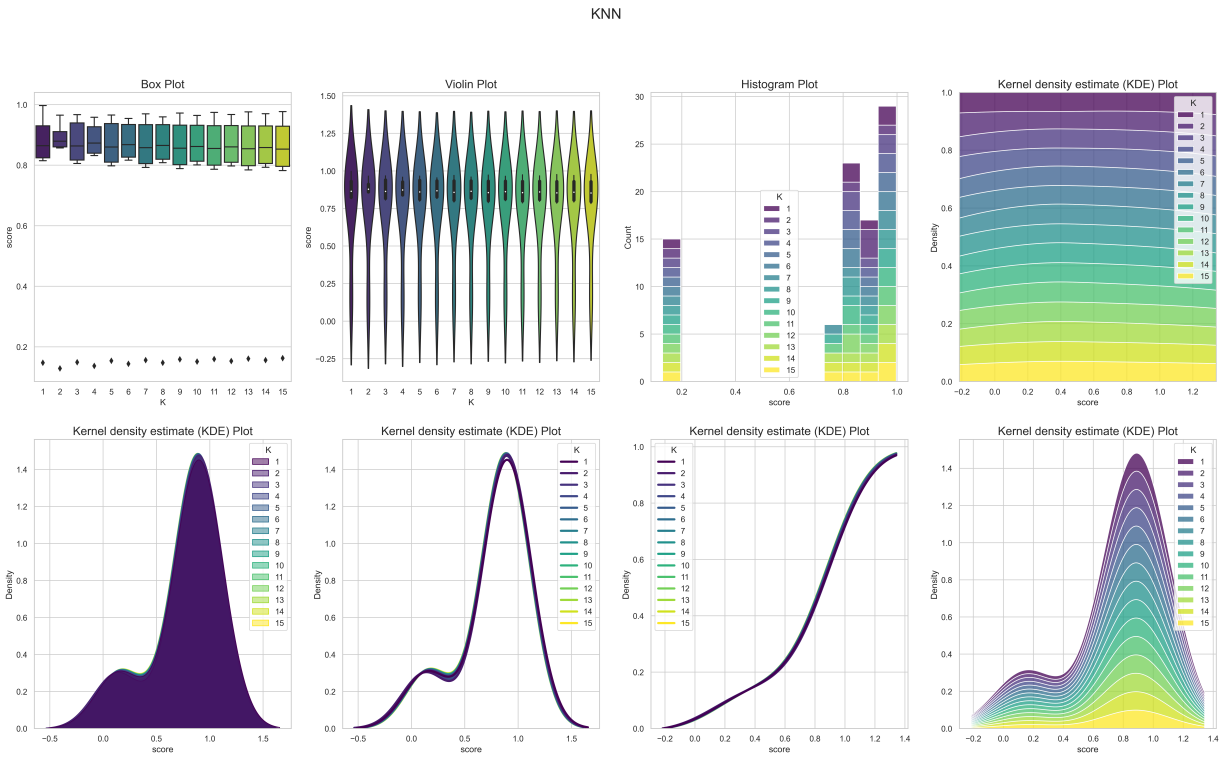


Figure 6.28: Distribution Plot for K 1 to 15

6.8 Performance Evaluation Summary

The Performance Evaluation Summary will recap the experiments done and the best result from each algorithm. The best-achieved result for each algorithm from the experimentations is presented in Table 6.15. The *Rank* represents the order in which algorithm performed the best, where Rank 1 indicates the best performing model, and Rank 6 indicates the worst performing model. The Parameter column shows the parameters used for that particular experiment. Finally, the runtime and runtimeGS show the manual search and Grid Search time it took to run each algorithm's experiments in total. The Table also included conventional statistics such as $mean, max, min,$ and $\text{diff}(min, max)$ for evaluation purposes. The percentage difference (increase/decrease) is calculated by Equation 6.6, where V_1 is the first value, and V_2 is the result of the percentage functioning on V_1.

$$\frac{|V_1 - V_2|}{\frac{(V1+V2)}{2}} \times 100 \tag{6.6}$$

Table 6.15: Best results from each algorithm

Rank	Algorithm	Parameters	Train Acc	Test Acc	F1	Precision	Recall	Runtime	Runtime GS
1	ANN	Epochs=20, 11 Layers	94.796	88.62	96.473	84.914	96.473	179499	323671
2	RF	criterion=entropy, n_estimators=200, maxDepth=none, minSS=2, minSL=1	99.817	87.399	89.603	82.094	98.626	19.3	49440
3	DT	criterion=entropy, minSL=20, minSS=25, maxdepth=55, maxfeatures=45, ccpAlpha=0.0000002	96.397	87.266	89.414	82.439	97.679	14	7680
4	KNN	K=2	96.521	87.11	88.702	85.703	91.919	580320	NaN
5	SVM	C=500	95.169	84.04	86.959	79.027	96.662	56526	763200
6	LR	maxIter=1600, C=10, Penalty=None, solver=lbfgs	93.759	81.835	85.632	75.849	98.312	815.9	8040
		Statistics							
		Mean	96.077	86.045	89.464	81.671	96.612	136199	230406
		Max	99.817	88.620	96.473	85.703	98.626	580320	763200
		Min	93.759	81.835	85.632	75.849	91.919	14.0	7680
		Difference (min, max)	6.46%	8.29%	12.66%	12.99%	7.30%	4145043%	9838%

6.8.1 Score Analysis

Observing Table 6.15 in combination with barplot 6.29, we can analyze the four performance metrics used in our thesis for the implemented algorithms.

- **Training Accuracy**: the average training accuracy across our six implemented algorithms equals 96.077%, with the best score of 99.817% for RF. The difference between the lowest and highest scores is 6.46%. This indicates a minimal error in training accuracy distributed among the algorithms.
- **Testing Accuracy**: the average testing accuracy across our six implemented algorithms equals 86.045%, with the best score of 88.62% for ANN. The difference between the lowest and highest scores is 8.29%. The combined results indicated that the algorithms perform well on our NDPF and output great results on unseen data.
- **Precision**: the average precision score for our implementation resulted in 81.671%, the best precision score of 81.671% for KNN, and the worst of 75.849%. The difference between the best and worst precision scores is 12.99%. The precision results indicate that the algorithms return more relevant feedback than irrelevant.
- **Recall**: the average recall score for our implementation resulted in 96.612%, with the best score of 98.626% for RF and the worst score of 91.919% for KNN. The difference between the worst and best scores is 7.30%. The results for recall indicate that our models predict a high amount of actual positives, meaning the results are relevant.
- **F1**: the average F1-Score for our models resulted in 89.464%, with the highest score of 96.473% for ANN and the lowest score of 85.632% for LR. The difference between the worst and best scores is 12.66%. The results indicate a balanced correlation between precision and recall, making our results valid.

The F1-Score is instrumental when comparing different ML algorithms on the same dataset. For example, a higher F1-Score indicates reasonably high recall and precision scores. Observing barplot 6.29, we can notice that the difference between the minimum and maximum values for recall and precision is relatively high. Hence F1 is an acceptable middle ground metric to rely on when such events occur to determine whether the results are valid.

Distribution plot 6.30 shows the distribution of the scores for each algorithm combined. For example, a score of 1 (100%) indicates a perfect prediction for any given performance metric in our thesis. Hence, the distribution plot 6.30 displays the ranges and how well the metrics are distributed among each algorithm. We can witness that, i.e., the combined distribution of the scores for ANN deviates minimally, implying that most of the metrics performed relatively equal.

Furthermore, by observing Table 6.15 and barplot 6.29, we can conclude that ANN outperformed the five other algorithms with accuracy and F1-Score consideration.

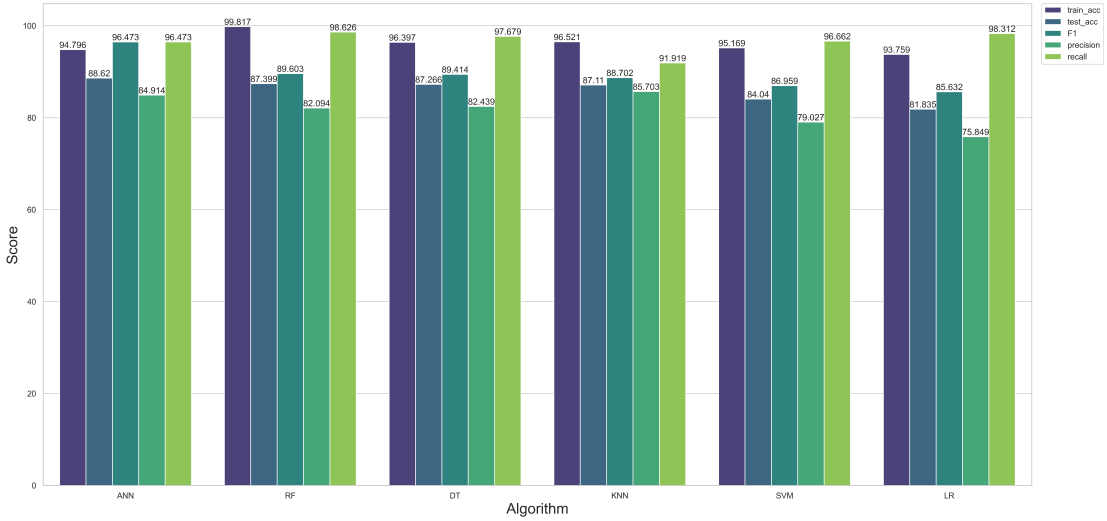


Figure 6.29: Comparison of Algorithm Performance

6.8.2 Algorithm Runtimes

The total run time from the manual- and grid search resulted in 547 hours (approximately 23 days) divided on multiple computers. The time estimate excludes the failed attempts like crashing and not being able to finish. Figure 6.31 illustrates the total time each algorithm consumed in seconds for running experiments, both with manual hyperparameter tuning and Grid Search.

By observing Table 6.15 and barplot 6.31, we can witness that DT has fastest runtime among the five algorithms, which resulted in a total runtime of 7694 s $\approx$ 2.137 h. Where we can also conclude the slowest longest runtime of 819726 s $\approx$ 227.701 h for SVM.

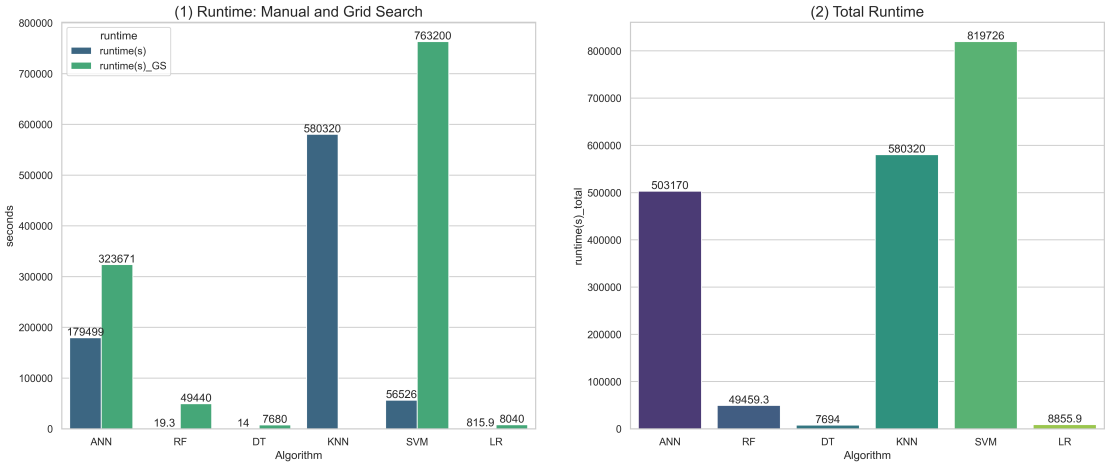


Figure 6.31: Total Runtime

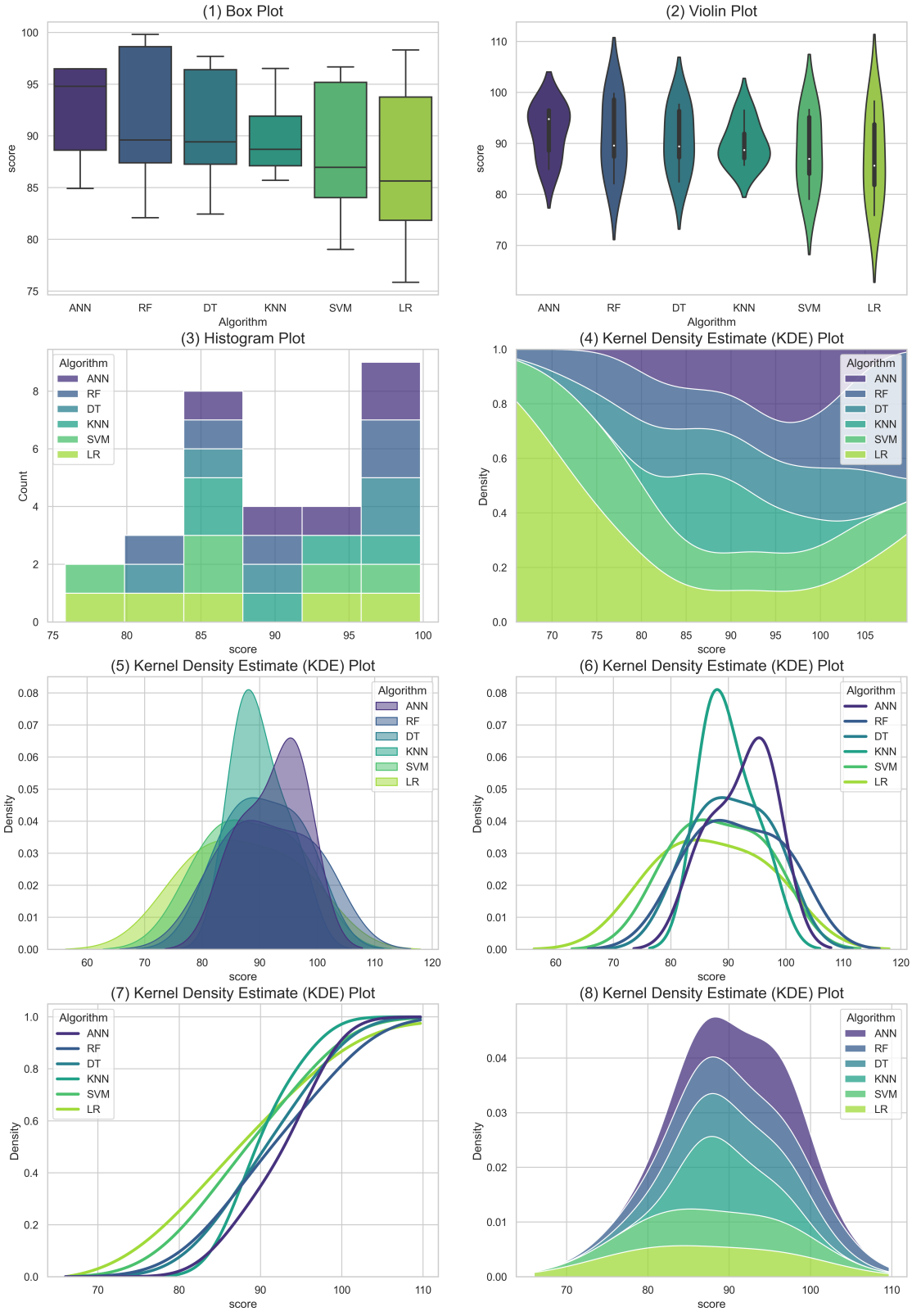


Figure 6.30: Distribution of Algorithm Performance

6.8.3 Performance Comparison with other Research

To compare our models' performance against other research, we searched for papers that utilized the same dataset (UNSW-NB15) and algorithms like us. We utilized databases like IEEE[3], Google Scholar[4], and ScienceDirect[5] for searching appropriate papers.

There are several limitations when searching for research to compare. Firstly, other research does not include multiple algorithms on their testing platform. Instead, they tend to focus on one or two algorithms at most. Second, the performance metrics included in their evaluation report do not have both training and testing accuracy. This makes it particularly challenging to know how their model performs on training and unseen testing data. Hence, there are some missing values in Table 6.17, where scores are represented in percentage (%). The statistical calculations we utilize such as $mean, max, min$, diff(min, max), diff(min, our), and diff(max, our), are explained in Table 6.16.

Table 6.16: Statistical Calculations

Statistical Calculation	**Description**
$mean$	Refers to the average score for the column
max	Refers to the maximum score from the column
min	Refers to the minimum score from the column
diff(min, max)	Refers to the difference in percentage for the lowest and highest score.
diff(min, our)	Refers to the difference in percentage for the smallest score and our score
diff(max, our)	Refers to the difference in percentage for the highest score and our score

Comparing the results of the papers for each algorithm, we utilize Table 6.17, 6.18, and Figure 6.32. When mentioning improvement, we refer to the the percentage difference (increase/decrease).

- **ANN**: in Table 6.17 and Figure 6.32. Our model scored a testing accuracy of 88.62%, proving it performed better than multiple papers from our comparison lot. Comparing our score with the lowest scoring paper for ANN, we can witness an improvement of 8.95% in testing accuracy and 15.73% in training accuracy. In contrast, Paper 5[37] outperforms our ANN model with a score of 89%, where the difference between our framework and their project is equal to 0.43%, which is minimal. The Table shows that Paper 5 only focuses on one algorithm, which is a crucial takeaway for our thesis.

- **SVM**: in Table 6.17 and Figure 6.32. Our model scored a testing accuracy of 84.04%, proving it scored better than multiple papers from the same table. Comparing our score with the lowest scoring paper for SVM, we can witness an improvement of 40.07% in testing accuracy and 34.08% in training accuracy, which is tremendous.

[3] `https://www.ieee.org`
[4] `https://scholar.google.com`
[5] `https://www.sciencedirect.com`

Regarding SVM, Paper 12[35] excels in testing accuracy with an improvement of 3.18% compared to ours.

- **DT**: in Table 6.17 and Figure 6.32. Our model scored a testing accuracy of 87.266%, proving it scored better than multiple papers from the same table. If we compare our score with the lowest scoring paper, we can notice an improvement of 36.35% in testing accuracy and 4.63% in training accuracy. The results show that even though other research papers score relatively high on training accuracy, their models do not have accurate predictions on unseen data. Paper 2[21] received a testing accuracy of 88.13%, and Paper 10[34] of 88.5%. Both papers outperformed our model for DT. Paper 10 has a score improvement of 1.39% compared to ours.

- **RF**: our second-best performing model, in Table 6.17 and Figure 6.32. Our model scored a testing accuracy of 87.399%, proving it scored better than multiple papers. Comparing our score with the lowest scoring paper, we notice an improvement of 36.56% in testing accuracy and 10.74% in training accuracy. Paper 10[34], and paper 12[35] outperforms our RF model with 87.7%, and 90.4% in testing accuracy, correspondingly.

- **KNN**: in Table 6.17 and Figure 6.32. Our model scored a testing accuracy of 87.11%. Comparing our score with the lowest scoring paper, we notice an improvement of 5.08% in testing accuracy and 13.33% in training accuracy. Our model received the highest score among the papers, with an improvement of 0.93% for testing accuracy, but for the training accuracy, Paper 2[21] received a slightly better improvement of 0.25%.

- **LR**: our weakest performing model in Table 6.17 and Figure 6.32. Our model received a testing accuracy of 81.835%. Comparing our score with the lowest scoring paper, we notice an improvement of 8.68% in testing accuracy and 0.58% in training accuracy. Regarding LR, our model got outperformed by Paper 6[31] and Paper 11[137]

Concluding, our Novel Data Preprocessing Framework (NDPF) and algorithm implementation with hyperparameter tuning show an average high performance across all the tested algorithms. Across all six algorithms used in our thesis, the average performance for testing accuracy is 86.045%, and 96.076% for training. The key takeaway for our performance summary comparing other papers is that our methods include several algorithms to compare and introduce several performance metrics to support our claims. In addition, our thesis uses the same Preprocessed Dataset from our NDPF to implement every algorithm presented. Furthermore, from our performance comparison, the difference between the highest-scoring research and ours differs in ±3.5%. Finally, analyzing the distribution plot 6.33, in general, ANN is the most robust algorithm for our chosen dataset, UNSW-NB15. The distribution of the scores does not deviate hugely compared to the other algorithms.

Table 6.17: Performance Comparison with other Papers

	ANN		SVM		KNN		LR		DT		RF	
	Train	*Test*	*Train*	*Test*	*Train*	*Test*	*Train*	*Test*	*Train*	*Test*	*Train*	*Test*
Our Thesis	**94.796**	**88.62**	**95.169**	**84.04**	**96.521**	**87.11**	**93.759**	**81.835**	**96.397**	**87.266**	**99.817**	**87.399**
Paper 1 [138]					93.71		93.23		94.2		95.43	
Paper 2 [21]	94.48	86.71	70.98	62.42	96.76	83.18	93.22	79.59	93.65	88.13		
Paper 3 [22]	81.91		86.8		86.31						90.14	
Paper 4 [30]	94.78		97.69								98.67	
Paper 5 [37]	96	89										
Paper 6 [31]		81.34						83.15		85.56		
Paper 7 [32]			87.05		85.17				92.13		90.34	
Paper 8 [33]							93.54		94.99		96.08	
Paper 9 [139]			92.28						95.82		97.49	
Paper 10 [34]		82.7				82.9		75.3		88.5		87.7
Paper 11 [137]		81.34						83.15		85.56		
Paper 12 [35]		81.91		86.8		86.31						90.14
Paper 13 [38]		88										
Paper 14 [36]				60						64		64
Paper 15 [39]		81.34						81.13		85.46		
Paper 16 [40]				81.5								
Paper 17 [41]	97											
Statistics												
Mean	93.161	84.551	88.328	74.952	91.694	84.875	93.437	80.693	94.531	83.497	95.424	82.310
Max	97.0	89.000	97.690	86.800	96.760	87.110	93.759	83.150	96.397	88.500	99.817	90.140
Min	81.340	81.340	70.980	60.000	85.170	82.900	93.220	75.300	92.130	64.000	90.140	64.000
Diff(min, max)	18.42%	9.42%	37.63%	44.67%	13.61%	5.08%	0.58%	10.42%	4.63%	38.28%	10.74%	40.84%
Diff(min, our)	15.73%	8.95%	34.08%	40.07%	13.33%	5.08%	0.58%	8.68%	4.63%	36.35%	10.74%	36.56%
Diff(max, our)	*-2.27%*	*-0.43%*	*-2.58%*	*-3.18%*	*-0.25%*	*0.93%*	*0.23%*	*-1.58%*	*0.60%*	*-1.39%*	*1.16%*	*-3.04%*

Table 6.18: Categorized Performance Comparison

	ANN			DT			RF	
	Training	*Testing*		*Training*	*Testing*		*Training*	*Testing*
Our Thesis	**94.796**	**88.62**	**Our Thesis**	**96.397**	**87.266**	**Our Thesis**	**99.817**	**87.399**
Paper 2 [21]	94.48	86.71	**Paper 1 [138]**	94.2		**Paper 1 [138]**	95.43	
Paper 3 [22]	81.91		**Paper 2 [21]**	93.65	88.13	**Paper 3 [22]**	90.14	
Paper 4 [30]	94.78		**Paper 6 [31]**		85.56	**Paper 4 [30]**	98.67	
Paper 5 [37]	96	89	**Paper 7 [32]**	92.13		**Paper 7 [32]**	90.34	
Paper 6 [31]		81.34	**Paper 8 [33]**	94.99		**Paper 8 [33]**	96.08	
Paper 10 [34]		82.7	**Paper 9 [139]**	95.82		**Paper 9 [139]**	97.49	
Paper 11 [137]		81.34	**Paper 10 [34]**		88.5	**Paper 10 [34]**		87.7
Paper 12 [35]		81.91	**Paper 11 [137]**		85.56	**Paper 12 [35]**		90.14
Paper 13 [38]		88	**Paper 14 [36]**		64	**Paper 14 [36]**		64
Paper 15 [39]		81.34	**Paper 15 [39]**		85.46			
Paper 17 [41]	97							
	SVM			**LR**			**KNN**	
	Training	*Testing*		*Training*	*Testing*		*Training*	*Testing*
Our Thesis	**95.169**	**84.04**	**Our Thesis**	**93.759**	**81.835**	**Our Thesis**	**96.521**	**87.11**
Paper 2 [21]	70.98	62.42	**Paper 1 [138]**	93.23		**Paper 1 [138]**	93.71	
Paper 3 [22]	86.8		**Paper 2 [21]**	93.22	79.59	**Paper 2 [21]**	96.76	83.18
Paper 4 [30]	97.69		**Paper 6 [31]**		83.15	**Paper 3 [22]**	86.31	
Paper 7 [32]	87.05		**Paper 8 [33]**	93.54		**Paper 7 [32]**	85.17	
Paper 9 [139]	92.28		**Paper 10 [34]**		75.3	**Paper 10 [34]**		82.9
Paper 12 [35]		86.8	**Paper 11 [137]**		83.15	**Paper 12 [35]**		86.31
Paper 14 [36]		60	**Paper 15 [39]**		81.13			
Paper 16 [40]		81.5						

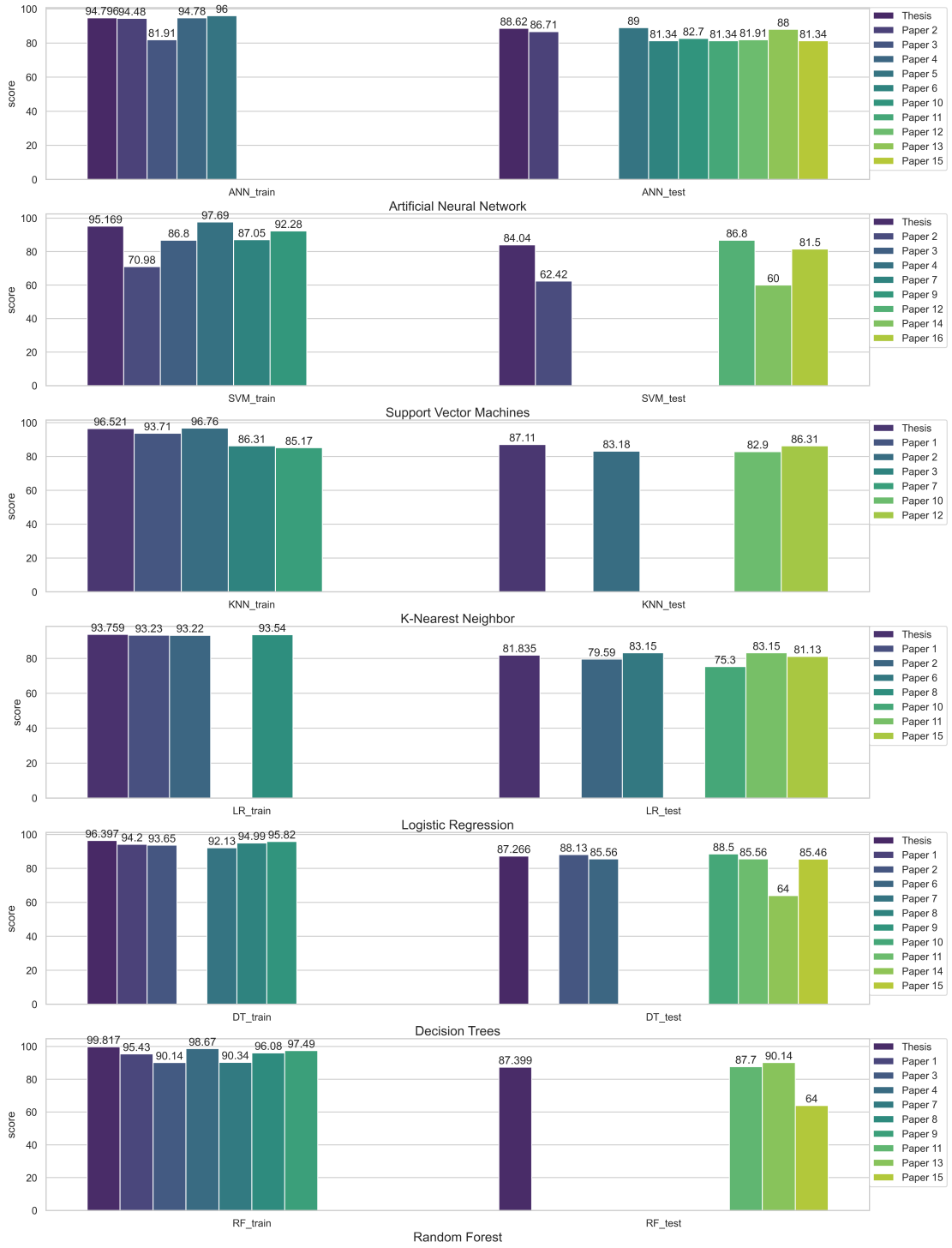


Figure 6.32: Barplot Comparing other Papers

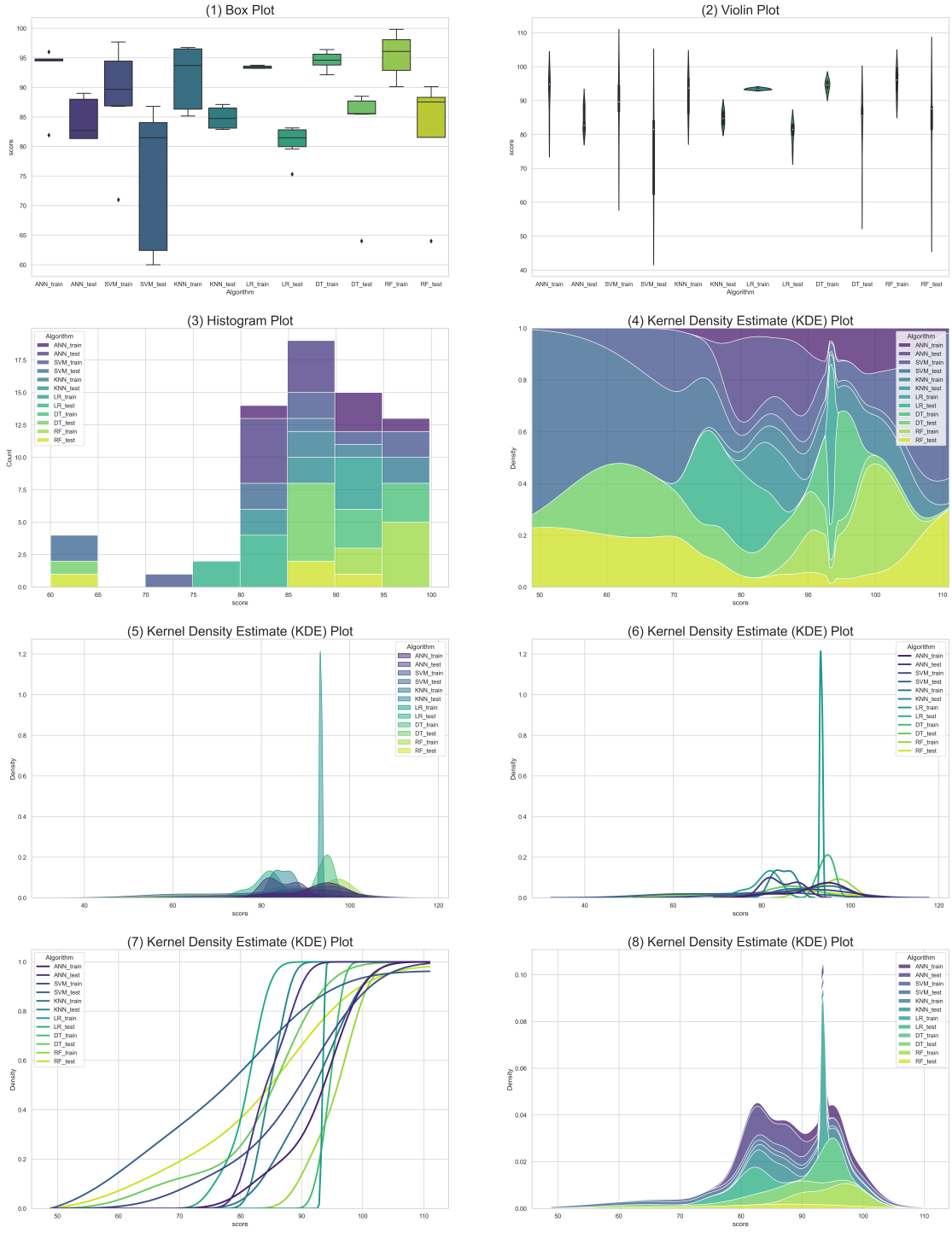


Figure 6.33: Distribution of Algorithm Performances across other Research

Chapter 7

Conclusion & Discussion

In this thesis, we aimed to address issues related to Network Intrusion Datasets with limitations regarding data preprocessing. Accordingly, the thesis objectives were presented in Section 1.5. As a result, we introduced the main elements of our work: the development of a Novel Data Preprocessing Framework (NDPF), the extensive hyperparameter tuning, and a comprehensive Performance Evaluation of our approach. The transformed UNSW-NB15 intrusion detection dataset aims to perform well on multiple ML algorithms. In order to evaluate the performance of the proposed framework, a comprehensive evaluation was done with six selected supervised ML algorithms: Artificial Neural Network, Support Vector Machine, K-Nearest Neighbor, Logistic Regression, Random Forest, and Decision Trees, and were compared against each other, and other research that used the same dataset. With the results from our experimentations, we can determine which algorithm performed best and if our proposed Novel Data Preprocessing Framework helped achieve satisfactory results. Furthermore, we will suggest some intriguing ideas for future work, which could improve the final result.

7.1 Contribution

When developing the NDPF, our chosen dataset, UNSW-NB15, went through an iterative process in several phases: Data Cleaning, Data Reduction, and Data Transformation. By applying these particular phases, we made the dataset optimized for ML algorithms. Data preprocessing is the essential step in any ML project, which also happens to be the most time-consuming process. The impact of preprocessing is enormous, where a slight difference in our methods can affect a given ML model either positively or negatively. The negative downside for a preprocessing method that does not consider the dataset can be the difference between an overfitting model and not.

Furthermore, we implemented the algorithms using Python inclusive libraries such as Scikit-learn and Keras. The ML models have an arbitrary amount of parameters to tune. Hence, our approach conducted an extensive hyperparameter tuning, divided into two methods, manual search and Grid Search. The manual search method implies that we set the parameters of our estimation, which requires domain knowledge within the algorithm

we hyperparameter tune. The manual search, as the name implies, is an iterative process. Further, the Grid Search algorithm is a technique to run a set of parameters until every combination is fulfilled, hence a brute-force method. Both approaches for hyperparameter tuning are exceptionally time-consuming, but they guarantee the best possible solution for our proposed method.

Finally, we conducted a Performance Analysis regarding our preprocessing framework, combined with hyperparameter tuning. The grounds for evaluation were based on several well-known performance metrics: Accuracy, Recall, Precision, and F1 Score. We documented and evaluated the results from numerous experiments for each algorithm and compared them against each other. In addition, a performance comparison was conducted against other research papers using the same dataset as the final analysis. It was to see whether our NDPF is effective. Finally, statistical analysis was performed to point out the differences in performance between our proposed framework with the existing solutions.

7.2 Concluding Remarks

Here we discuss the objectives stated in Chapter 1. Table 7.1 shows our objectives in addition to our accomplishment level.

Table 7.1: Thesis Accomplishment

Main Category	Purpose	Sub Category		Objective	Description	Accomplishment level
Background Analysis (B)	To review and determine appropriate dataset and methods for our thesis	**B1**		Dataset Selection	Select a suitable and well-balanced Network Intrusion Dataset.	*High*
		B2		Preprocessing methods	Review preprocessing methods available.	*High*
		B3		Performance results for other works	Review other works for chosen dataset.	*High*
		B4		Algorithm Review	Review popular Machine Learning algorithms.	*High*
		B5		Algorithm Selection	Select appropriate algorithms for selected dataset.	*High*
		B6		Development Environment	Select Development environment and programming language.	*High*
Novel Data Preprocessing Framework (P)	To create a well-performing dataset across multiple algorithms, by developing a Novel Preprocessing Framework	**P1**	***P1.1***	Data Cleaning	Remove missing and redundant data.	*High*
			P1.2	Feature Selection	Select the most optimal features.	*High*
			P1.3	Feature Engineering	Create or remove features manually based on existing data.	*High*
			P1.4	Data Transformation	Transform the dataset, and prepare for implementation.	*High*
		P2	***P2.1***	Feature Selection with algorithms	Automatic Feature Selection with RFE.	*Low*
			P2.2	Feature Selection, own method	Random Forest Classifier with 10-fold Cross-validation.	*High*
Algorithm Implementation (A)	Implement our chosen algorithms, and hyperparameter tune our models to maximize our results.	**A1**	***A1.1***	Implementation of ANN	Implement the algorithms for the preprocessed Dataset. Hyperparameter tuning for maximizing results.	*High*
			A1.2	Implementation of SVM		*High*
			A1.3	Implementation of KNN		*High*
			A1.4	Implementation of LR		*High*
			A1.5	Implementation of DT		*High*
			A1.6	Implementation of RF		*High*
	Develop a novel algorithm for our dataset that performs better than existing ones.	**A2**		Develop a Novel Algorithm	Develop a Novel algorithm, for our preprocessed dataset.	*Unaccomplished*
Evaluation (E)	Present evidence that our method performs well across multiple algorithms, compared to other existing works.	**E1**	***E1.1***	Comparable results for ANN	Result Comparison with other works evaluating the corresponding algorithms.	*Medium-High*
			E1.2	Comparable results for SVM		*Medium-High*
			E1.3	Comparable results for KNN		*High*
			E1.4	Comparable results for LR		*Medium-High*
			E1.5	Comparable results for DT		*Medium-High*
			E1.6	Comparable results for RF		*Medium-High*
		E2		Visualize Results	Visualize the results in an informative way.	*High*

7.2.1 Background Analysis (B)

The objective of Background Analysis (B) was to gather research and information regarding five subjects: dataset, preprocessing, algorithm, and development environment.

- **Sub-Category B1:** Data selection aimed to select the most appropriate network intrusion detection dataset for our project. After numerous comparisons and reviews with other network intrusion datasets such as KDD-Cup 1999, NSL-KDD, Gure-KDD, CICIDS2017, and BoT-IoT, we decided to select the UNSW-NB15 dataset, which is widely used by the scientific community. In addition, we chose the dataset based on the number of research already done, making it easier for us to compare results. We can therefore conclude that the accomplishment level of Sub-category B1 equals *High*, shown in Table 7.1

- **Sub-Category B2:** We reviewed several preprocessing techniques to achieve a high-performing dataset. Based on our results, our review process can be claimed an accomplishment level of *High*.

- **Sub-Category B3:** Referring to the literature study, we have established an overview of existing research on our dataset and topic, hence accomplishment level, *High*.

- **Sub-Categories B4 and B5:** According to Chapter 4 describing the different ML algorithms, we have chosen the most appropriate ones based on community favorites and popularity: ANN, SVM, LR, KNN, DT, and RF. Choosing popular ML algorithms implies that the existing research on the topic usually consists of those algorithms, making performance comparison more manageable. Hence, our selection of ML algorithms is the most suitable, resulting in accomplishment level, *High*.

- **Sub-Category B6:** According to Section 5.1, we have chosen the most suitable tools for our project. Visual Studio Code is an excellent IDE for developing any code. Furthermore, combining the Visual Studio Code IDE with Python and Anaconda package manager makes the project flow considerably more practical. Hence, the selection of tools and libraries results accomplishment level of, *High*.

7.2.2 Novel Data Preprocessing Framework (P)

The objective of our proposed Novel Data Preprocessing Framework (P) was to develop a framework with essential data preprocessing techniques, enhanced to a point where the dataset performs well on multiple algorithms.

- **Sub-Category P1:** We successfully made a Novel Data Preprocessing Framework (NDPF) that performed well across multiple algorithms. The NDPF includes several techniques such as data cleaning, feature selection, feature engineering, and data transformation. According to our results, we can claim the level of accomplishment as *High*.

- **Sub-Category P2:** Firstly, feature selection with RFE returned features of high importance, resulting in inaccurate performance. Therefore, feature selection with RFE receives an accomplishment level of *Low*. On the other hand, our feature

selection method successfully extracted the essential features from the UNSW-NB15 dataset, resulting in an accomplishment level of *High*.

7.2.3 Algorithm Implementation (A1)

Algorithm Implementation (A) aimed for two objectives: (A1) Implementing existing ML algorithms for our NDPF, and (A2) Develop a Novel algorithm utilizing our NDPF.

- **Sub-Category A1:** We implemented all of the six algorithms from our algorithm selection (B5) using existing libraries from Scikit-learn. We hyperparameter tuned our models during implementation and results gathering with manual search and Grid Search. Our performance evaluation (Chapter 6) was based on these exact implementations and their results. Hence, the implementation of existing ML algorithms receives an accomplishment level of *High*.

- **Sub-Category A2**: Our thesis aimed to develop a novel algorithm, which in combination with our NDPF, outperforms existing research using the UNSW-NB15 dataset. Unfortunately, due to time constraints, we could not deliver this objective. There are several time-consuming aspects to our current experimentations and analysis methods. Accordingly, we have referred to some ideas for future work (Section 7.3) regarding the further development of our thesis. Hence, we can conclude that the accomplishment level for this particular objective is *Unaccomplished*.

7.2.4 Evaluation (E)

- **Sub-Category E1:** We presented in detail the experimentations' results, which were firstly compared among the six selected algorithms. Finally, we offered detailed evidence that our NDPF, combined with Hyperparameter tuning, performed well-balanced compared to other research using the same dataset, UNSW-NB15. The results from our experimentation and comparison show that the output is not necessarily the best performing, but across multiple algorithms, our NDPF satisfies the overall outcomes. Hence, this objective receives an accomplishment level of *Medium-High*. The results are intriguing and open up possibilities for other research to append our NDFP.

- **Sub-Category E2:** The performance analysis for each section was visualized adequately to support our claims. The visualization was confined to tables, distribution plots, bar plots, line plots, and others.

The majority of the objective stated by us in Chapter 1 receives an accomplishment level of *High*. There are some aspects to consider when referring to our *Unaccomplished* objective A2 regarding the Novel algorithm development. As mentioned earlier, some intriguing ideas and concepts will be referred to and shortly explained in Section 7.3, *Future Work*

7.3 Future Work

One of the goals of this thesis was to develop a novel algorithm, but due to time constraints mentioned earlier, we did not get around to this. However, we have come up with a few possible ways to enhance a few of the algorithms for future work. The ideas involve making algorithms less computational heavy or performing better. We will list the ideas here and briefly explain the thoughts behind the ideas.

Feature Selection

There are several ways to execute feature selection. There are possibilities to rank the features by utilizing an algorithm for feature selection. Further, we feed the rank data into an ML algorithm, which focuses on training with the essential features. As a result, this can reduce the training complexity by spending less time on features that do not contribute much information.

Single Learners in Ensemble algorithm

In an ensemble algorithm, we can enhance the single learners. The weak learners become more of an asset, resulting in a more robust ensemble algorithm. For example, in RF, we can assign each tree a weight depending on how strong the weak learner can select the appropriate tree.

Improved Hyperparameter tuning

There are already several hyperparameter tuning methods that might be more useful than Grid Search. For example, Gu et al.[140] adopted a more recent and improved tuning method, namely FOA-F-SVM. While this method is not entirely new, it can be an excellent way to improve our work, especially for SVM, a computational heavy, brute-force algorithm.

Decision Tree Pruning

Pruning work on DT is essential to avoid overfitting and to improve accuracy. A possible pruning method that gives estimators such as accuracy, tree complexity, attribute selectors, and other metrics a score. This score will be used to easier prune a tree by knowing which of the leaves it can take away. The ones with the lowest score will be taken away first. The score is calculated by using Bayes minimum risk method. This method can prove to be faster than traditional cost complexity pruning [141].

ANN Optimizers

One way to enhance ANN is to explore the different optimizers. For example, there are many different optimizers for ANN[1] and several ways to use them. Ramesh et al.[142]

[1] `https://keras.io/api/optimizers/`

proposed a Text-to-Image generation application, where the critical takeaway which is relevant for our thesis is the utilization of multiple ANN optimizers during model training. It is also possible to look into enhancing a specific optimizer explicitly suited for the task at hand.

SVM Kernels

The kernels in SVM have a basis for improvements, which can enhance the algorithm. However, when experimenting with SVM, many kernels have not been tested in our thesis. At the same time, the kernels are how the algorithm performs the calculations to map the data to higher dimensions. Therefore, as a result of optimizing the kernels, both accuracy and run time might be improved.

KNN

KNN utilizes a brute-force method to calculate the distance to the neighbor of a given point. However, executing the algorithm with brute force has a high time complexity for training[143]. Therefore, exploring different methods to calculate the distance between points without visiting every neighbor might be a way to enhance the algorithm [144].

Multi-Class Classification

The UNSW-NB15 dataset also includes nine different attack categories in correlation with the label (0 or 1), meaning the dataset can be utilized for Multi-Class Classification. Multi-Class Classification has more than two output classes, contrary to Binary Classification with one output. With Multi-Class Classification, the algorithm can determine its kind of attack, in addition to the label. Future work can involve testing our NDPF for Multi-Class Classification for an even more comprehensive Performance Evaluation, which can weigh our claims more.

Apply our NDPF to other datasets

Applying our NDPF to other intrusion detection datasets, we can determine whether the limitations of data preprocessing and intrusion detection datasets still apply. For future work, we can practice the following datasets: KDD-Cup 1999, NSL-KDD, Gure-KDD, BoT-IoT, and CICIDS2017.

7.3.1 Concluding Statement for Future Work

Future work has many possible ways to increase and enhance the performance for different aspects of our thesis. For example, Hyperparameter tuning and even slight changes in parameters can result in simpler time-complexity. The above-mentioned ideas are only a tiny fraction of possible enhancements and methods to explore in the future

7.4 Final Conclusion

Comparing our results with those from the other 17 other papers, using the same dataset, UNSW-NB15, we could analyze the results tables, bar plots, and distribution plots, that ANN excels in performance in every aspect. In our experimentation, ANN received an accuracy of 88.62% and an F1 score of 96.47%, making the algorithm recommendable for any project. To finalize our findings compared with other papers, our chosen ML models, ANN, KNN, DT, RF, LR, and SVM, performed comparably to the other research. Nevertheless, other papers on several algorithms slightly outperformed ours, with a percentage difference of $\pm 3.5\%$. However, the critical takeaway is that the other research mainly focuses on one or two algorithms, where they centralize the preprocessing of their dataset to those specific algorithms. On the other hand, our NDPF transforms the dataset to perform in a balanced manner across multiple algorithms, making it remarkably stable. Therefore, we can conclude our Novel Data Preprocessing Framework (NDPF) as a success and make recommendations for other future studies.

Publications

1. Sajepan Gnanasivam, Daniel Tveter and Nga Dinh. "Anomaly Intrusion Detection for Data Network Using K-Nearest Neighbor", *submitted to IEEE International Symposium on Local and Metropolitan Area Network*, 2022.

2. Sajepan Gnanasivam, Daniel Tveter and Nga Dinh. "Performance Evaluation for Anomaly Intrusion Detection in Data Networks", *submitted to IEEE Global Communications Conference*, 2022.

3. Sajepan Gnanasivam, Daniel Tveter and Nga Dinh. "Performance Assessment for Intrusion Detection using Supervised Machine Learning Algorithms on the UNSW-NB15 dataset", *to be submitted to Journal of Big Data*, 2022.

Performance Evaluation for Anomaly Intrusion Detection in Data Networks

Sajepan Gnanasivam*, Daniel Tveter*, Nga Dinh*
* Department of Computer Science and Communications
Faculty of Computer Science, Engineering and Economics
Østfold University College (HiOF), Norway
{sajepan.gnanasivam, daniel.tveter, thi.t.dinh}@hiof.no

***Abstract*—The number of computers connected to the Internet and hostile attacks on data networks increases daily. Therefore, anomaly detection, which finds network abnormalities using supervised classification approaches, is essential. Thus, this paper focuses on anomaly detection for data networks and uses the UNSW-NB15 dataset, including over 82,000 samples and 49 features. Due to their various origins, most real-world datasets are prone to missing, inconsistent, and noisy data. Implementing data mining algorithms to this noisy data would produce poor results. Consequently, data preprocessing is critical for improving overall performance. Our preprocessing method includes several necessary procedures: data- cleaning, -transformation, and feature selection (dimensionality reduction and feature engineering). This paper conducts performance evaluation by extensive experimentation on several data mining techniques and presents a novel preprocessing method. Data mining techniques such as Artificial Neural Network (ANN), Support Vector Machines (SVM), Logistic Regression (LR), and Decision Tree (DT) are implemented, performing binary classification. The experiments maximize accuracy to discover the right set of hyper-parameters. The results from the experiments indicate that ANN excels with a training accuracy of 98.58%, DT with 96.02%, SVM with 95.1%, and LR with 93.76%. The achieved results are comparable and, in many cases, better than existing works due to our preprocessing framework.**

***Index Terms*—Anomaly Detection, Machine Learning, Supervised Methods, Feature Engineering, Data Networks**

I. Introduction

The massive growth in cyber threats, coupled with modern organizations' reliance on the stability and effectiveness of their IT infrastructure, has prompted a shift in mentality. As a result, priorities are altering as the downtime increases for systems within information technology. Intrusions, such as brute force, denial of service, or even penetration within a network, are the most common threat to a network's security. Furthermore, cyber criminals are violently attempting to interrupt network connections, get illegal access to essential data, and then steal, distort, or damage essential data structures. Hence, a dynamic approach to detect and prevent such intrusions is required. It is a critical challenge in the field of computer network security that must be addressed. In light of this, anomaly detection has been hugely popular [1]. The goal of network anomaly detection is to find network abnormalities, where supervised classification approaches have yielded several successes through many types of research.

The difficulty of discovering unusual patterns in network traffic that do not conform to predicted typical behavior is referred to as anomaly-based intrusion detection in networks. In many applications, these nonconforming patterns are referred to as anomalies, aberrations, outliers, or surprises [2]. Efforts have been undertaken over the last three decades to develop automated systems for detecting network irregularities [3].

Anomaly detection in data networks is a vital and rapidly evolving field of study since our everyday is becoming more and more data-driven. With more data and information available than ever before, it is critical to analyze and evaluate it properly [4]. In data network, anomaly detection has a wide range of applications, including credit card fraud detection, cyber security intrusion detection, and military observation of adversary activity. An unusual traffic pattern in a computer network, for example, could indicate that a hacked computer is transferring sensitive data to an unauthorized site [2]. According to Cisco's Annual Internet Report, the frequency of breaches and the number of records exposed to each breach are increasing. Between 2018 and 2019, attacks between 100 Gbps and 400 Gbps increased by 776%, and the total number of DDoS attacks will double from 7.9 million in 2018 to 15.4 million by 2023 [5]. Due to the importance of anomaly detection, this paper focuses on detecting aberrations in computer networking.

Several machine learning (ML) algorithms were applied in performance analysis on the UNSW-NB15 dataset [6]. The proposed method included a feature selection technique utilizing XGboost. As a final result with binary classification, ANN received a training accuracy of 94.49%, LR with 93.76, SVM with 70.98%, and DT with 93.65% In another study [1], network abnormality in the UNSW-NB15 dataset is detected using a time-based statistical analysis. According to the findings, the labeled dataset may be utilized to accurately train both supervised and unsupervised ML systems. In addition, the authors discovered that the suggested model might be applied to detect network anomalies that have not been addressed previously in network traffic by extracting features from the network packet using unsupervised learning. Furthermore, in [7], the authors present a framework to detect malicious behavior in cloud environments also based on UNSW-NB15 dataset.DT, RF, Naive Bayes, KNN, LR, and SVM are among the ML algorithms used by the authors. The results reveal that with feature selection, DT received an accuracy of 96.77%, LR with 74.73% and SVM with 68.05%.

This paper focuses on anomaly intrusion detection on the UNSW-NB15 dataset while considering binary classification. In the experimental phase, we focused on four different supervised ML algorithms: ANN, SVM, DT, and LR. Our extensive experiments show that the implemented ML algorithms performed better or were comparable to other research done on the same dataset [6]. However, prior to any experiments or hyperparameter tuning, the dataset had to undergo a preprocessing method to transform the dataset into a readable format. With the preprocessed dataset, we could do hyperparameter tuning. First, each algorithm went through two methods, either manual tuning or an algorithm that goes through a set parameter grid to find the optimal parameters for each algorithm. Then, the algorithms are evaluated based on the four metrics: accuracy, recall, precision, and F1-score, where accuracy is the primary metric for our evaluation compared to other researchers.

The main contributions of this paper can be summarized as follows: 1) Propose a novel data preprocessing framework. Our preprocessing framework includes several necessary procedures: data- cleaning, -reduction (feature selection and feature engineering), and -transformation; 2) Hyperparameter tuning to select the best parameters for each ML algorithm; 3) We investigate the performance of four supervised ML algorithms through extensive experiments with a total runtime of 272 hours (11.5 days). With our novel data preprocessing framework and hyperparameter tuning, our results are superior to other research with the same dataset and ML algorithms [6] for binary classification.

II. Dataset

Dataset is the most crucial aspect of a network security analysis. The amount of the data also impacts the performance of ML algorithms. Selecting the most significant features from the input data can simplify the modeling process and result in faster and more accurate detection rates. Frequently, datasets contain multiple useless, redundant features, which is disadvantageous to the accuracy of the results. In light of this, the UNSW-NB15 dataset is selected. The dataset has also been widely used in many researches [8]–[10].

The UNSW-NB15 dataset was created by researchers from the Australian Center for Cyber Security (ACCS) in 2015 [10], [11] with 49 features. The UNSW-NB15 dataset is more sophisticated and reflective of current attack and normal network traffic, making it suitable for evaluating network intrusion techniques. The number of records in the UNSW-NB15 dataset is 2,540,044 which is stored in four .csv files represented in Table I. Each of the dataset contains attack and normal records. A training set and a testing set were created from the data collection, named UNSW-NB15_training-set.csv and UNSW-NB15_testing-set.csv, respectively. The training set has 82,332 records while the testing set contains 175,341 records including attack and normal records. Do note that the training and testing sets are reversed.

TABLE I
Dataset overview

File name	File Size	Records	Features
UNSWNB15_1.csv	161.2 MB	700000	49
UNSWNB15_2.csv	157.6 MB	700000	49
UNSWNB15_3.csv	147.4 MB	700000	49
UNSWNB15_4.csv	91.3 MB	440044	49
UNSW_NB15_testing-set.csv	31.5 MB	175341	45
UNSW_NB15_training-set.csv	15.0 MB	82332	45

III. Data Preprocessing

Before any ML algorithm is applied to a dataset, the data has to be preprocessed so that the processed data is in the format that the model can comprehend (fig 1). The UNSW-NB15 contains categorical and non-similar scale features that need to be preprocessed in order to make the dataset compliant with a ML algorithm [1]. To ensure that the selected dataset is converted into a suitable one, we perform data preprocessing with *data cleaning*, *data transformation*, and *data reduction*.

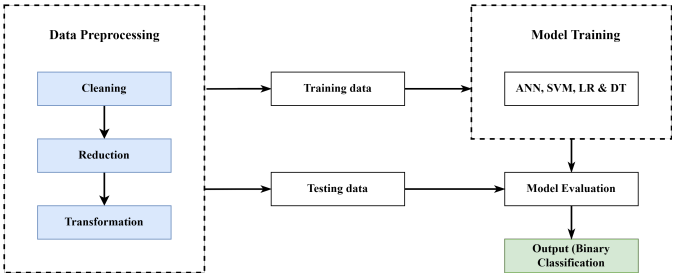


Fig. 1. The complete framework

A. Data Cleaning

Data Cleaning aims to fix the problem with incomplete, inconsistent, and noisy data. But first, the data that is missing or irrelevant needs to be processed. *Missing data points* is a common attribute that many datasets have. There are several approaches to solve this problem (1) *Duplicating values* rows or columns that occur more than twice in the same dataset. Dropping a duplicate row or column and keeping the first instance is one of the solutions to this problem. The reason for doing this is not to make the specific data point take advantage of or be biased. (2) *Dropping rows and columns* that includes a not a number (NaN) field or an empty object (NULL) values. (3) *Estimating missing values* is used if a small percentage of the data is lost. Data imputation techniques such as mean, median, or mode values are the most common approach to apprehend this problem.

After a thorough analysis, the UNSW-NB15 does not exhibit any missing values to remove or perform data imputation.

B. Data Transformation

The second phase of data preprocessing is Data Transformation which aims to convert the data into a known format readable to a ML model. Data normalization transforms numerical columns into a standard scale (-1.0 to 1.0 or 0.0

to 1.0). This process transforms the data columns without distorting the differences in the data range, meaning it will maintain the dynamic range of the dataset.

1) Reversed Datasets: The UNSW-NB15 dataset has the training and testing set samples reversed, caused by an error from the publishers. This means that the testing set samples are double the training set size. We promptly fix the issue by running an if statement on both sets and checking whether the training set has fewer or larger samples than the testing set. The testing set contains 175,341 rows and 45 columns opposed to the training set, corresponding to 82,332 rows and 45 columns.

2) Feature Engineering: The process of changing the raw data into features that better describe the underlying problem to predictive models, resulting in enhanced model accuracy on unseen data, is known as feature engineering. Feature engineering depends on domain knowledge in a specific field to select and transform features from a dataset. It is evident that many features can share requirements characteristics. The procedure entails a combination of data analysis, rule-of-thumb use, and judgment.

The feature engineering process starts with analyzing the various values for each feature for the dataset. Then, by running `train['state'].value_counts` from the pandas library, we output the row count for each distinct value in a feature, in this case, 'state'. Observing the table shows that the row counts for the states 'FIN', 'INT', 'CON' and 'REQ' are more significant than the others. Therefore, the states which have lower row counts are renamed to 'Others'.

The '-' value is present under the 'service' feature, which equals 47,153. This value is converted to 'Others' since the dataset description informs us that the value is a service that is not frequently used. The same method is replicated for the 'service' and 'proto' features. We also observe the row counts for those features and renamed the values to 'Others'. The purpose of this procedure is to reduce the complexity of the model.

3) Categorical Columns: Our ML algorithm cannot process Categorical data points, meaning the transformation of the data points into a readable format is essential for the model to run without errors. Therefore, we drop the 'label' feature since our model performs binary classification. Further, to make the categorical features readable by the model, one-hot encoding needed to be implemented. Therefore, we transform each categorical value into a new categorical column and assign a binary value of 1 or 0 to those columns using one-hot encoding. A binary vector is used to represent each integer value. The `pd_dummies()` function from the Pandas library accomplished exactly this assignment.

4) Normalization: The UNSW-NB15 dataset has a broad dynamic range of values, indicating that normalization needs to be applied on the numerical columns. The core concept of standardizing/normalizing is to individually delegate $\mu = 0$, and $\sigma = 1$ for the features of X. `Standardscaler()` will independently normalize the features so that each column/feature will produce $\mu = 0$, and $\sigma = 1$. x equals to the observation, μ is the mean and σ is the standard deviation. The mathematical representation for Standard Deviation is given in equation 1

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{1}$$

C. Data Reduction

Dealing with a large amount of data for analysis comes with difficulty. Data reduction aims to improve storage efficiency and lower data storage and analysis expenses. The concept of *Dimensionality Reduction* is to reduce the number of features or dimensionality. Data reduction is, in our case, done in the *Feature Selection*. But firstly, we analyze the dataset by looking at the correlation matrix.

1) Correlation: Correlation is the degree to which two variables are linearly related is expressed by correlation. If there is an extensive dataset with many columns, displaying the correlation matrix as a heatmap is a rapid approach to assess relationships between columns. When a dependent and independent variable has a high correlation value, the independent variable affects the output.

The Pearson r correlation is the most extensively utilized statistical approach for determining the degree of linearly related variables. The Pearson correlation between any two variables x, y can be calculated by using Equation (2), where n is number of observations, and i-denotes the i-th observation.

$$r_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}} \tag{2}$$

Fig. 2 shows a graphical representation of the correlation matrix for several features with coefficient values given by a color scale.

2) Feature Selection: Feature selection is one of the core concepts in ML. This process hugely impacts the performance of any ML model. When developing a predictive model, the primary objective is to reduce the number of input variables. The number of input variables included in a model's training stage dramatically influences the performance, meaning feature selection and data cleaning should be the first and most crucial step of the developing phase.

Reducing the number of input variables lessens the computational cost and, in many cases, enhances the performance of a given model. The process has two given approaches:

1) Automatically determine input variables using various algorithms
2) Manually choose input variables to remove from the model.

The benefits of reducing the number of input variables are (1) reduce overfitting i.e, the dataset will become less redundant, which in hand will make fewer opportunities to make a decision based on noise, (2) accuracy improvement i.e, training data will be less misleading, which will improve accuracy, and (3) training time reduction i.e, the complexity of the algorithm decreases.

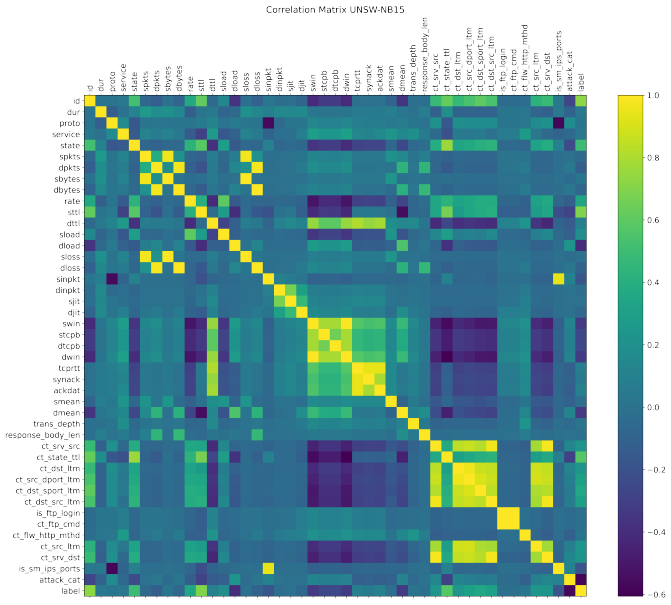


Fig. 2. Correlation plot for UNSW-NB15 Training dataset

3) Method 1 - Recursive Feature Elimination (RFE): RFE is a well-known feature selection algorithm used to identify which features in a training dataset are more significant in predicting the target variable. RFE works by searching for a subset of features in the training dataset, starting with all features and successfully eliminating them until the targeted number remains. The method can be accomplished by re-fitting the model using the provided ML technique, ranking features by relevance, excluding the least important features, and fitting the model again. This procedure is executed until only a certain amount of features are left. Features are evaluated by using either a ML model or a statistical technique. In our work, RFE has concluded with the top features and has left us with the following features to drop: `attack_cat`, `id`, `response_body_len`, `spkts`, `ct_flw_http_mthd`, `trans_depth`, `dwin`, `ct_ftp_cmd` and `is_ftp_login`, which we tend to drop.

IV. Supervised Machine Learning Algorithms

A. Artificial Neural Network (ANN)

Artificial Neural Networks (ANN), often known as neural networks, are a subset of ML algorithms that are at the heart of deep learning algorithms. A neural network's core concept is to imitate a large number of tightly interconnected brain cells inside a computer so that it can learn new things, identify patterns, and make decisions in a human-like approach.

ANNs include an input layer, one or more hidden layers, and an output layer, each of which links to the layers on either side. They range from a few dozen to hundreds, thousands, or even millions of artificial neurons known as units. The connections between two units are represented by a weight, which might be positive (if one unit excites another) or negative (if one unit repels another). The greater the impact of one unit on another, the higher the weight.

Neural networks use training data to learn and increase their accuracy over time. However, once these learning algorithms have been fine-tuned for accuracy, they become effective tools in computer science and artificial intelligence, allowing us to categorize high-density data quickly. For example, human specialists' manual identification, speech recognition, or picture recognition tasks can take minutes rather than hours.

B. Support Vector Machines (SVM)

Support vector machine is one of the most robust ML algorithms. Its also highly flexible and can work with both regression and classification problems. SVM's main objective is to separate any given dataset into different classes. The way it separates the dataset is by finding the optimal hyperplane using the support vectors. Some advantage of SVM is that it got multiple kernels which got different use for different issues. SVM is also quite effective for data with high-dimensional input features. One downside to SVM is that it requires extensive tuning to the parameters, significantly when the input dimension exceeds the number of samples the dataset has, quickly leading to overfitting.

C. Logistic Regression (LR)

Logistic regression (LR) can do both binary and multiclass classification, which is the primary use for this ML algorithm. Even though the algorithm has regression in its name, it is rarely used for regression tasks. When used for multiclass classification, it uses the one versus all concept to learn. Logistic regression uses either the sigmoid function(5) or a variation. This is applied to the linear model (3) and gives an output between 0 and 1. The output is the probability of belonging to the class labeled as 1, where the value 0.5 is referred to as the soft boundary to the algorithm. The distance between the boundary increases as the output goes towards 1 or 0. The equation given for logistic regression (4), where y is a process of σ

$$y = b + w_1x_1 + w_2x_2 + ... + w_nx_n \tag{3}$$

$$\sigma(y) = \frac{1}{1 + e^{-y}} \tag{4}$$

$$\sigma(K) = \frac{1}{1 + e^{-k}}c \tag{5}$$

D. Decision tree (DT)

Decision trees are a supervised learning algorithm method used for regression and classification problems. The objective is to make a model that can predict a value based on a target variable. To make the model, it first needs to make rules based on the training features. The model is easy to understand and interpret, and multiple tools can be used to visualize it.

V. Performance Evaluation

The preprocessing and the build of the ML algorithms is executed on a computer with the following specifications: operating system is macOS Catalina(10.15.4) with Intel Core i9 9880H, @2.3 GHz CPU, AMD Radeon Pro 5500M with 4GB GDDR6 graphics card, and 32GB DDR4 memory @2667 MHz. Jupyter lab and Visual Studio code have been our development environment with Anaconda distribution. Python version 3.8.8 has been used as the only programming language to analyze and develop our project.

A. Performance Metrics

We look at four different measurements when analyzing the algorithms in this paper; Accuracy (ACC), Recall (RCL), Precision (PRC), and F1-score. Out of those four, we aimed to maximize ACC for each algorithm.

1) Accuracy: Based on the input or training data, ACC is used to recognize patterns and correlations among variables in a dataset. ACC is defined by (6)

$$ACC = \frac{TP + TN}{(TP + FP + FN + TN)} \tag{6}$$

In (6), True Positive (TP) is where the actual class is positive and the predictor class is positive. In our case, if the outcome label is 1 (attacked) and the predicted value is 1. True Negative (TN), i.e, outcome label is 0 (not attacked) on the predicted and 0 on the actual label class. False Negative (FN) and False Positive (FP) are when the predicted and the actual class contradict each other. FP is when the label output is 0 and the predicted value is 1. On the other hand, FN is when the label output is 1 and the predicted value is 0.

2) Recall: RCL offers information on the percentage of the real anomalies that have been classified. RCL provides information about the percentage of the true anomalies that were classified. This equation determines this measure (7).

$$RCL = \frac{TP}{(TP + FN)} \tag{7}$$

3) Precision: PRC is the number of correct predictions of the positive observations compared to the total amount of predicted positive observations. So in our case, it is how many attacks that were labeled as an attack are actually attacked (8).

$$PRC = \frac{TP}{(TP + FP)} \tag{8}$$

4) F1-Score: The F1-score metric incorporates both Recall and Precision scores and uses a mean value approach to estimate the overall performance of the anomaly detection algorithm. This is used to calculate the equation (9).

$$F1 = \frac{2 \times (Recall \times Precision)}{Precision + Recall} \tag{9}$$

B. Artificial Neural Network (ANN)

A total number of seven experiments were conducted for ANN. Every experiment was run for 200 epochs, with a different number of layers and neurons, which is presented in Table II. The experiments' best accuracy of 98.5% is achieved with nine layers. The experiments have a different number of layers and neurons. The first experiment has three layers with a neuron structure of 50(input), 25(hidden), 1(output). The second experiment has four layers with neurons structure: 53-36-21-1. The third experiment has four layers with neurons structure: 500-250-100-1. The fourth experiment has five layers with neurons structure: 100-75-50-25-1. The fifth experiment has seven layers with neurons structure: 200-150-125-100-50-25-1. The sixth experiment has nine layers with neurons structure: 200-175-150-125-100-75-50-25-1. Finally, the seventh experiment included eleven layers with neurons structure: 300-250-200-175-150-125-100-75-50-25-1.

TABLE II
ANN Experiments

L	Epoch	Train Acc	Test Acc	F1	PRC	RCL	Time(s)
3	170	0.955	0.872	0.975	0.959	0.975	941.0
4	200	0.959	0.864	0.978	0.963	0.978	1232.6
4	190	0.986	0.849	0.993	0.986	0.993	2603.3
5	200	0.972	0.862	0.981	0.978	0.981	1905.8
7	200	0.984	0.857	0.991	0.985	0.991	2462.5
9	200	0.985	0.861	0.990	0.989	0.990	3041.1
11	140	0.978	0.856	0.989	0.980	0.989	3294.7

C. Support Vector Machine (SVM)

Using the preprocessed dataset with the default parameters on SVM, we received an accuracy of 81.57% on binary classification. Running an extensive search on the C value determines the trade-off between misclassification loss and maximizing the margin. At C equal to 500, gamma equal to scale, with RBF kernel, we got an accuracy of 84.03%. Table III shows the results of the best results, and some other experiments from the tuning phase.

TABLE III
SVM experiments

Kernel	rbf	poly	poly	rbf
C	500	500	250	250
Gamma	scale	scale	auto	auto
Train_ACC	0.951	0.947	0.945	0.948
Test_ACC	0.840	0.830	0.834	0.832
F1	0.869	0.861	0.867	0.864
PRC	0.790	0.781	0.775	0.780
RCL	0.966	0.960	0.983	0.968
Time(s)	2590	5300	1850	1530

D. Logistic Regression (LR)

LR resulted in an accuracy of 81.4% on the testing dataset, with a max iteration of 1000 on the preprocessed dataset. To get higher accuracy with LR, we had to find optimal values for the parameters: C, max_iter, penalty, and solver. Using the grid search algorithm, we got the following values shown in

Table IV. With the tuned parameters, the accuracy increased to 81.8%.

TABLE IV
LR EXPERIMENT

Max_iter	100	10	25	50
C	100	10	25	50
Penalty	l2	none	l2	l2
Solver	newton-cg	lbfgs	saga	sag
Train_ACC	0.937	0.930	0.931	0.932
Test_ACC	0.818	0.807	0.805	0.807
F1	0.856	0.847	0.846	0.848
PRC	0.758	0.753	0.747	0.749
RCL	0.983	0.967	0.976	0.976
Time(s)	0.1	0.1	0.1	5.3

E. Decision Tree

The DT algorithm went through several steps. The first model resulted in an accuracy of 86.3% on the testing set with no parameter tuning. Then, to increase the accuracy of the DT model, post pruning techniques were applied. The post pruning provided the DT model with slightly better accuracy of 86.4%. In the last experiment, we used the grid search algorithm. The results are shown in Table V with 87.28% being the best.

TABLE V
DT EXPERIMENT

Criterion	gini	gini	entropy	entropy
Min_samples_leaf	20	10	10	15
Min_samples_split	14	10	10	15
Max_features	16	15	15	20
Ccp_alpha	2×10^{-17}	2×10^{-7}	2×10^{-7}	2×10^{-9}
Train_ACC	0.960	0.967	0.968	0.964
Test_ACC	0.867	0.868	0.872	0.872
F1	0.980	0.889	0.893	0.894
PRC	0.818	0.824	0.825	0.822
RCL	0.976	0.966	0.973	0.980
Time(s)	0.6	0.6	0.6	0.9

In table VI we compare the results from our project with another study [6], which proves that our novel preprocessing framework is better and stable across multiple ML algorithms.

TABLE VI
RESULTS COMPARISON

Algorithm	**train_acc**	**train_acc [6]**	**test_acc**	**test_acc [6]**
ANN	0.985	0.944	0.861	0.942
DT	0.968	0.936	0.872	0.933
SVM	0.951	0.709	0.840	0.706
LR	0.937	0.932	0.818	0.928

VI. CONCLUSION

To receive accurate predictions with binary classification, this research investigated a novel framework for data preprocessing on the UNSW-NB15 dataset, in combination with several ML algorithms. The original UNSW-NB15 dataset was first preprocessed with data cleaning, transformation, and reduction. The comprehensive preprocessing has undergone an iterative process, intending to improve and get better results.

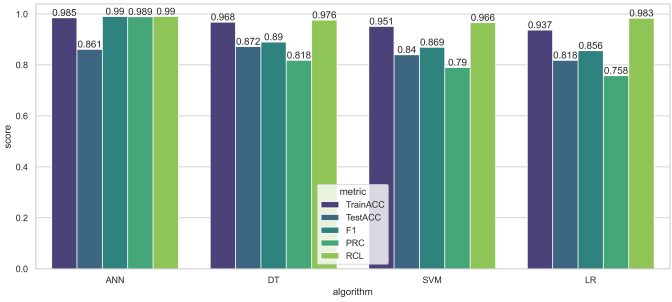


Fig. 3. Performance Comparison

The data preprocessing and hyperparameter tuning have been our project's utmost critical work. The ML algorithms were run extensively with manual and automated hyperparameter tuning to find the most optimal parameters for our preprocessed dataset to create satisfactory results. The results (fig 3) indicate that our novel preprocessing method is comparable and, in many cases, superior correlated to other studies. Therefore, our project aims to support other researchers who intend to use our preprocessing methods and hyperparameter tuning settings for future studies.

REFERENCES

[1] D. Sujana, S. Hegde, C. Pankaj, S. Suresh, P. Ramakrishna *et al.*, "Temporal based network packet anomaly detection using machine learning," in *2021 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*. IEEE, 2021, pp. 1–6.

[2] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: methods, systems and tools," *Ieee communications surveys & tutorials*, vol. 16, no. 1, pp. 303–336, 2013.

[3] M. Tavallaee, N. Stakhanova, and A. A. Ghorbani, "Toward credible evaluation of anomaly-based intrusion-detection methods," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 40, no. 5, pp. 516–524, 2010.

[4] J. Long, F. Fang, and H. Luo, "A survey of machine learning-based iot intrusion detection techniques," in *2021 IEEE 6th International Conference on Smart Cloud (SmartCloud)*. IEEE, 2021, pp. 7–12.

[5] B. Wang, C. Wang, W. Huang, Y. Song, and X. Qin, "A survey and taxonomy on task offloading for edge-cloud computing," *IEEE Access*, vol. 8, pp. 186 080–186 101, 2020.

[6] S. M. Kasongo and Y. Sun, "Performance analysis of intrusion detection systems using a feature selection method on the unsw-nb15 dataset," *Journal of Big Data*, vol. 7, no. 1, pp. 1–20, 2020.

[7] P. Jha and A. Sharma, "Framework to analyze malicious behaviour in cloud environment using machine learning techniques," in *2021 International Conference on Computer Communication and Informatics (ICCCI)*. IEEE, 2021, pp. 1–12.

[8] L. Zhiqiang, G. Mohi-Ud-Din, L. Bing, L. Jianchao, Z. Ye, and L. Zhijun, "Modeling network intrusion detection system using feed-forward neural network using unsw-nb15 dataset," in *2019 IEEE 7th International Conference on Smart Energy Grid Engineering (SEGE)*. IEEE, 2019, pp. 299–303.

[9] M. Hammad, W. El-medany, and Y. Ismail, "Intrusion detection system using feature selection with clustering and classification machine learning algorithms on the unsw-nb15 dataset," in *2020 International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT)*. IEEE, 2020, pp. 1–6.

[10] M. Zeeshan, Q. Riaz, M. A. Bilal, M. K. Shahzad, H. Jabeen, S. A. Haider, and A. Rahim, "Protocol-based deep intrusion detection for dos and ddos attacks using unsw-nb15 and bot-iot data-sets," *IEEE Access*, vol. 10, pp. 2269–2283, 2021.

[11] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 military communications and information systems conference (MilCIS)*. IEEE, 2015, pp. 1–6.

Anomaly Intrusion Detection for Data Networks Using K-Nearest Neighbor

Sajepan Gnanasivam*, Daniel Tveter*, Nga Dinh*
* Department of Computer Science and Communications.
Faculty of Computer Science, Engineering and Economics.
Østfold University College (HiOF), Norway
{sajepan.gnanasivam, daniel.tveter, thi.t.dinh}@hiof.no

***Abstract*—With the explosively increasing number of computers connected to the Internet and the rise in hostile attacks on data networks, it is crucial than ever to protect the computer networks. In light of this, anomaly detection has been hugely popular. The goal of anomaly detection is to find network abnormalities where supervised classification approaches have yielded several successes through many types of research.**

This paper aims to evaluate and analyze the performance of K-Nearest Neighbor (KNN) binary classification with Euclidean- and Minkowski distance calculations for network intrusions. The UNSW-NB15 dataset, including over 82,000 samples and 49 features, is used in the paper. In addition, preprocessing in which the dataset has undergone different phases such as data-cleaning, reduction, and transformation is performed. Finally, the performance of KNN is evaluated through extensive experiments to discover the right K-value for the preprocessed dataset. The results show that the accuracy, precision, and F1-measure decrease as the K-value increases. Nevertheless, the highest score at K equal to 2 with an accuracy of 87.107% is achieved.

***Index Terms*—Anomaly Detection, KNN, Machine Learning, Supervised Methods, Data Network.**

I. Introduction

The massive growth in cyber threats, coupled with modern organizations' reliance on the stability and effectiveness of their IT infrastructure, has prompted a shift in mentality. As a result, priorities are altering as the downtime increases for systems within information technology. Intrusions, such as brute force, denial of service, or even penetration within a network, are the most common threat to a network's security. Furthermore, cyber criminals are violently attempting to interrupt network connections, get illegal access to essential data, and then steal, distort, or damage essential data structures. Hence, a dynamic approach to detect and prevent such intrusions is required. It is a critical challenge in the field of computer network security that must be addressed. In light of this, anomaly detection has been hugely popular due to these issues [1]. The goal of network anomaly detection is to find network abnormalities, where supervised classification approaches have yielded several successes through many types of research.

Anomaly detection in data networks is a vital and rapidly evolving field of study since our everyday is becoming more and more data-driven. With more data and information available than ever before, it is critical to analyze and evaluate it properly [2]. In data network, anomaly detection has a wide range of applications, including credit card fraud detection, cyber security intrusion detection, and military observation of adversary activity. An unusual traffic pattern in a computer network, for example, could indicate that a hacked computer is transferring sensitive data to an unauthorized site [3]. According to Cisco's Annual Internet Report, the frequency of breaches and the number of records exposed to each breach are increasing. Between 2018 and 2019, attacks between 100 Gbps and 400 Gbps increased by 776%, and the total number of DDoS attacks will double from 7.9 million in 2018 to 15.4 million by 2023 [4]. Due to the importance of anomaly detection, this paper focuses on detecting aberrations in computer networking.

Machine Learning has been efficiently used in anomaly detection. An ensemble feature selection method [5] is applied on the TON_IoT dataset [6]. Here, the authors use multiple classifiers to evaluate the performance of each model. In particular, Random Forest has the best accuracy of 99.23% and a F1-score of 98.90% among the seven classifiers whereas Decision Tree has the quickest execution time. K-Nearest Neighbor (KNN) achieves a accuracy of 83.41% and F1-score of 74.81%. In another study [1], network abnormality in the UNSW-NB15 dataset is detected using a time-based statistical analysis. According to the findings, the labeled dataset may be utilized to accurately train both supervised and unsupervised machine learning systems. In addition, the authors discovered that the suggested model might be applied to detect network anomalies that have not been addressed previously in network traffic by extracting features from the network packet using unsupervised learning. Furthermore, in [7], the authors present a framework to detect malicious behavior in cloud environments also based on UNSW-NB15 dataset. Decision Tree, Random Forest, Naive Bayes, KNN, Linear Regression, and SVM are among the machine learning algorithms used by the authors. The results reveal that with feature selection, Random Forest surpassed all other models with accuracy, sensitivity, and specificity of 97.84%, 98.04%, and 97.68%, respectively. Decision trees and KNN came second and third, respectively, with 96.77% and 80.45% accuracy.

Machine Learning algorithms such as Decision Tree (DT) [8] and Support Vector Machines (SVM) [9] have been used extensively for the UNSW-NB15 dataset. However, there is not enough attention for KNN, whereas KNN has better pre-

diction accuracy on other datasets [10], [11]. Hence, this paper proposes an anomaly intrusion detection framework using the KNN based on the UNSW-NB15 dataset. The framework includes a data preprocessing phase and an implementation phase to evaluate the performance of KNN with different K-values. In the data preprocessing phase, our works on data cleaning, data transformation, and data reduction are documented. In the implementation phase, we implement and perform binary classification with a KNN model with extensive different K-values on our preprocessed dataset to determine the algorithm's performance. The models are evaluated concerning the accuracy, recall, precision, and F1-score performance metrics for K-values up to 200. Based on our evaluation, the best setting of K value is recommended.

The rest of the paper is organized as follows. Section II is the introduction of the UNSW-NB15 dataset which is selected to conduct the analysis and experiments. Section III presents our comprehensive works on data preprocessing including data cleaning, data transformation, and data reduction. The KNN Machine Learning algorithm is described in section IV. In section V, we evaluate the performance of KNN on our preprocessed UNSW-NB15 dataset. Finally, section VI concludes this paper.

II. Dataset

Dataset is the most crucial aspect of a network security analysis. The amount of the data also impacts the performance of Machine Learning algorithms. Selecting the most significant features from the input data can simplify the modeling process and result in faster and more accurate detection rates. Frequently, datasets contain multiple useless, redundant features, which is disadvantageous to the accuracy of the results. In light of this, the UNSW-NB15 dataset is selected. The dataset has also been widely used in many researches [12]–[14].

The UNSW-NB15 dataset was created by researchers from the Australian Center for Cyber Security (ACCS) in 2015 [14], [15] with 49 features. The UNSW-NB15 dataset is more sophisticated and reflective of current attack and normal network traffic, making it suitable for evaluating network intrusion techniques. The number of records in the UNSW-NB15 dataset is 2,540,044 which is stored in four .csv files represented in Table I. Each of the dataset contains attack and normal records. A training set and a testing set were created from the data collection, named UNSW-NB15_training-set.csv and UNSW-NB15_testing-set.csv, respectively. The training set has 82,332 records while the testing set contains 175,341 records including attack and normal records. Do note that the training and testing sets are reversed.

TABLE I
Dataset overview

File name	File Size	Records	Features
UNSWNB15_1.csv	161.2 MB	700000	49
UNSWNB15_2.csv	157.6 MB	700000	49
UNSWNB15_3.csv	147.4 MB	700000	49
UNSWNB15_4.csv	91.3 MB	440044	49
UNSW_NB15_testing-set.csv	31.5 MB	175341	45
UNSW_NB15_training-set.csv	15.0 MB	82332	45

III. Data Preprocessing

Before any Machine Learning algorithm is applied to a dataset, the data has to be preprocessed so that the processed data is in the format that the model can comprehend. The UNSW-NB15 contains categorical and non-similar scale features that need to be preprocessed in order to make the dataset compliant with a Machine Learning algorithm [1]. To ensure that the selected dataset is converted into a suitable one, we perform data preprocessing with *data cleaning*, *data transformation*, and *data reduction*.

A. Data Cleaning

Data Cleaning aims to fix the problem with incomplete, inconsistent, and noisy data. But first, the data that is missing or irrelevant needs to be processed.

Missing data points is a common attribute that many datasets have. There are several approaches to solve this problem:

1) Duplicating values rows or columns that occur more than twice in the same dataset. Dropping a duplicate row or column and keeping the first instance is one of the solutions to this problem. The reason for doing this is not to make the specific data point take advantage of or be biased.
2) Dropping rows and columns that includes a not a number (NaN) field or an empty object (NULL) values.
3) Estimating missing values is used if a small percentage of the data is lost. Data imputation techniques such as mean, median, or mode values are the most common approach to apprehend this problem.

After a thorough analysis, the UNSW-NB15 does not exhibit any missing values to remove or perform data imputation.

B. Data Transformation

The second phase of data preprocessing is Data Transformation which aims to convert the data into a known format readable to a machine learning model. Data normalization transforms numerical columns into a standard scale (-1.0 to 1.0 or 0.0 to 1.0). This process transforms the data columns without distorting the differences in the data range, meaning it will maintain the dynamic range of the dataset.

1) Reversed Datasets: The UNSW-NB15 dataset has the training and testing set samples reversed, caused by an error from the publishers. This means that the testing set samples are double the training set size. We promptly fix the issue by running an if statement on both sets and checking whether the training set has fewer or larger samples than the testing set. The testing set contains 175,341 rows and 45 columns opposed to the training set, corresponding to 82,332 rows and 45 columns.

2) Feature Engineering: The process of changing the raw data into features that better describe the underlying problem to predictive models, resulting in enhanced model accuracy on unseen data, is known as feature engineering. Feature engineering depends on domain knowledge in a specific field to select and transform features from a dataset. It is evident that many features can share requirements characteristics. The procedure entails a combination of data analysis, rule-of-thumb use, and judgment.

The feature engineering process starts with analyzing the various values for each feature for the dataset. Then, by running `train['state'].value_counts` from the pandas library, we output the row count for each distinct value in a feature, in this case, 'state'. Observing the table shows that the row counts for the states 'FIN', 'INT', 'CON' and 'REQ' are more significant than the others. Therefore, the states which have lower row counts are renamed to 'Others'.

The '-' value is present under the 'service' feature, which equals 47,153. This value is converted to 'Others' since the dataset description informs us that the value is a service that is not frequently used. The same method is replicated for the 'service' and 'proto' features. We also observe the row counts for those features and renamed the values to 'Others'. The purpose of this procedure is to reduce the complexity of the model.

3) Categorical Columns: Our machine learning algorithm cannot process Categorical data points, meaning the transformation of the data points into a readable format is essential for the model to run without errors. Therefore, we drop the 'label' feature since our model performs binary classification. Further, to make the categorical features readable by the model, one-hot encoding needed to be implemented. Therefore, we transform each categorical value into a new categorical column and assign a binary value of 1 or 0 to those columns using one-hot encoding. A binary vector is used to represent each integer value. The `pd_dummies()` function from the Pandas library accomplished exactly this assignment.

4) Normalization: The UNSW-NB15 dataset has a broad dynamic range of values, indicating that normalization needs to be applied on the numerical columns. The core concept of standardizing/normalizing is to individually delegate $\mu = 0$, and $\sigma = 1$ for the features of X. `Standardscaler()` will independently normalize the features so that each column/feature will produce $\mu = 0$, and $\sigma = 1$. x equals to the observation, μ is the mean and σ is the standard deviation. The equation for standard deviation (3) is achievable by taking the equation for standardization (1) and combining it with the mean for each sample (2).

$$z = \frac{x - \mu}{\sigma} \tag{1}$$

$$\mu = \frac{1}{N}\sum_{i=1}^{N}(x_i) \tag{2}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{3}$$

C. Data Reduction

Dealing with a large amount of data for analysis comes with difficulty. Data reduction aims to improve storage efficiency and lower data storage and analysis expenses. The concept of *Dimensionality Reduction* is to reduce the number of features or dimensionality. Data reduction is, in our case, done in the *Feature Selection*. But firstly, we analyze the dataset by looking at the correlation matrix.

1) Correlation: Correlation is the degree to which two variables are linearly related is expressed by correlation. If there is an extensive dataset with many columns, displaying the correlation matrix as a heatmap is a rapid approach to assess relationships between columns. When a dependent and independent variable has a high correlation value, the independent variable affects the output.

The Pearson r correlation is the most extensively utilized statistical approach for determining the degree of linearly related variables. The Pearson correlation between any two variables x, y can be calculated by using Equation (4), where n is number of observations, and i-denotes the i-th observation.

$$r_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}} \tag{4}$$

Fig. 1 shows a graphical representation of the correlation matrix for several features with coefficient values given by a color scale.

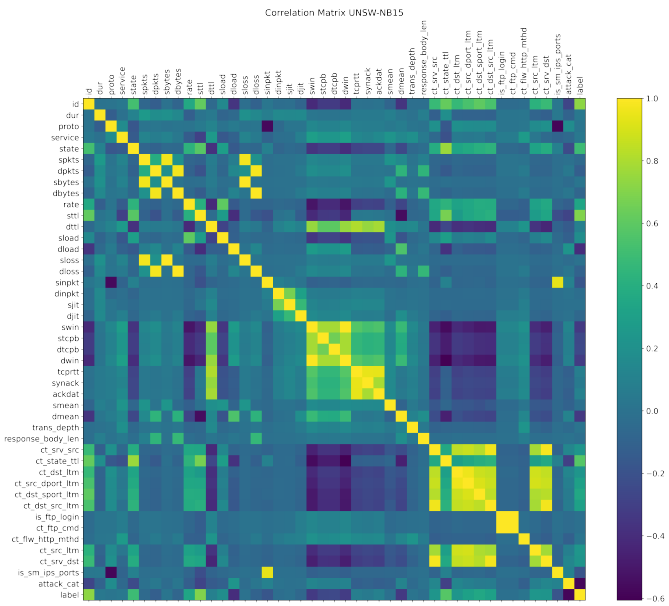


Fig. 1. Correlation plot for UNSW-NB15 Training dataset

2) Feature Selection: Feature selection is one of the core concepts in Machine Learning. This process hugely impacts the performance of any Machine Learning model. When developing a predictive model, the primary objective is to reduce the number of input variables. The number of input variables included in a model's training stage dramatically influences the performance, meaning feature selection and data cleaning should be the first and most crucial step of the developing phase.

Reducing the number of input variables lessens the computational cost and, in many cases, enhances the performance of a given model. The process has two given approaches:

1) Automatically determine input variables using various algorithms
2) Manually choose input variables to remove from the model.

The benefits of reducing the number of input variables are (1) reduce overfitting i.e, the dataset will become less redundant, which in hand will make fewer opportunities to make a decision based on noise, (2) accuracy improvement i.e, training data will be less misleading, which will improve accuracy, and (3) training time reduction i.e, the complexity of the algorithm decreases.

3) Method 1 - Recursive Feature Elimination (RFE): RFE is a well-known feature selection algorithm used to identify which features in a training dataset are more significant in predicting the target variable. RFE works by searching for a subset of features in the training dataset, starting with all features and successfully eliminating them until the targeted number remains. The method can be accomplished by re-fitting the model using the provided machine learning technique, ranking features by relevance, excluding the least important features, and fitting the model again. This procedure is executed until only a certain amount of features are left. Features are evaluated by using either a machine learning model or a statistical technique. In our work, RFE has concluded with the top features and has left us with the following features to drop: `attack_cat`, `id`, `response_body_len`, `spkts`, `ct_flw_http_mthd`, `trans_depth`, `dwin`, `ct_ftp_cmd` and `is_ftp_login`, which we tend to drop.

IV. K-Nearest Neighbor Algorithm

The supervised Machine Learning KNN algorithm can address both classification and regression tasks [16], [17]. The KNN assumes that similar data points exist in close proximity meaning that there are near to each other. Distance metrics such as *Euclidean distance, Hamming distance, Manhattan distance* and *Minkowski distance* are utilized to estimate the distance between points for locating the closest similar points. In this paper, we use Euclidean Distance, calculated by (5) and Minkoswki Distance, calculated by (6), for KNN implementation. In equation (5), x_i and y_i are the coordinators of the i^{th} featured element, respectively.

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{5}$$

$$d(x,y) = (\sum_{i=1}^{n}|x_i - y_i|^p)^{1/p} \tag{6}$$

The main advantages of KNN: (1) The algorithm is simple and straightforward to implement, (2) There is no need to create a model, tweak parameters, or make any further assumptions, (3) The algorithm is highly adaptable. It has classification, regression, and search capabilities. One of the most significant disadvantages is that the algorithm gets much slower as the number of input/independent -variables increases. Meaning the volume of data has a significant impact on the runtime for the algorithm. Thus, KNN is not suitable for applications where real-time data output is crucial. However, with proficient computer resources, KNN can be very useful for identifying similar objects. The main steps of the KNN is shown in **Algorithm 1.**

Algorithm 1 K-Nearest Neighbor (KNN) Algorithm

Input: K-value, dataframe in
Output: Class (Attack or Non-attack) out
Initialization :
1: Load preprocessed splitted training and testing datasets.
LOOP Process
2: **for** $training\_data\_points$ **do**
3: Calculate Euclidean Distance given by equation (5)
4: Sort calculated distances in ascending order
5: Get top K rows from the sorted array
6: Assign a class to the training data point, based on majority voting
7: **end for**
8: **return** $predicted\_class$

V. Performance Evaluations

The preprocessing and the build of the KNN model is executed on a computer with the following specifications: operating system is macOS Catalina(10.15.4) with Intel Core i9 9880H, @2.3 GHz CPU, AMD Radeon Pro 5500M with 4GB GDDR6 graphics card, and 32GB DDR4 memory @2667 MHz. Jupyter lab and Visual Studio code have been our development environment with anaconda distribution. Python version 3.8.8 has been used as the only programming language to analyze and develop our project.

The performance of the KNN is affected by the chosen K value. KNN is run numerous times with different values of K to determine which K was the most suitable for our dataset. To evaluate the performance of each model, we have used different metrics to cover the various aspects of the scoring output. The metrics are *Accuracy*, *Recall*, *Precision*, *F1* and *Running time*. The scoring is calculated in combinations of True-Positive (TP), True-Negative (TN), False-Positive (FP), and False-Negative (FN).

1) *Accuracy:* The accuracy is utilized to recognize correlations and patterns between variables in a dataset based

on the input or training data. This metric is calculated by equation (7).

$$Accuracy = \frac{TP + TN}{(TP + TN + FP + FN)} \quad (7)$$

2) *Recall:* Recall gives us information about what portion of true anomalies were classified. This metric is determined by equation (8).

$$Recall = \frac{TP}{(TP + FN)} \quad (8)$$

3) *Precision:* Precision represents the ratio of true positives to all predicted positives. In our case, how many of the output labeled as attacks, are actual attacks? This metric is determined by equation (9).

$$Precision = \frac{TP}{(TP + FP)} \quad (9)$$

4) *F1 measure:* F1 measure uses a mean value method to determines the overall performance of the anomaly detection model by incorporating both Recall and Precision scores. This matric is calculated by equation (10).

$$F1 - measure = \frac{2 \times (Recall \times Precision)}{Precision \times Recall} \quad (10)$$

Table (II) presents the result of our experiment with the values of K from 1 to 15. Our predictions become less stable as we increase the value of K to 15. Thus, the predictions become less stable due to majority voting and averaging, and less accurate. The average accuracy for K(1, 15) equals 84.899%, where the highest score was 87.107% for K(2) and the lowest score of 84.755% for K(15). Fig. 2 represents a bar plot of the different K values, ranging from 1 to 15. We can see which K value performs best by analyzing the plot's accuracy, precision, recall, and F1-measure.

TABLE II
EXPERIMENTS (K 1 TO 15)

K	Accuracy	F1	Precision	Recall	Runtime
1	0.8528	0.8764	0.8149	0.9479	3m 39s
2	0.8710	0.8870	0.8570	0.9192	3m 51s
3	0.8508	0.8764	0.8057	0.9606	4m 3s
4	0.8633	0.8835	0.8321	0.9417	4m 17s
5	0.8462	0.8736	0.7977	0.9655	4m 15s
6	0.8561	0.8794	0.8167	0.9525	3m 50s
7	0.8436	0.8723	0.7927	0.9696	3m 49s
8	0.8525	0.8775	0.8083	0.9596	3m 52s
9	0.8414	0.8710	0.7887	0.9724	3m 50s
10	0.8487	0.8753	0.8012	0.9646	3m 52s
11	0.8404	0.8705	0.7866	0.9743	3m 54s
12	0.8467	0.8742	0.7974	0.9673	3m 50s
13	0.8392	0.8698	0.7845	0.9760	3m 53s
14	0.8440	0.8726	0.7928	0.9701	3m 52s
15	0.8375	0.8688	0.7821	0.9771	3m 51s

The decrease in accuracy as the K value increases ought up to a question, which refers to the next experiment i.e, investigate whether a higher K value results in better or worse model accuracy.

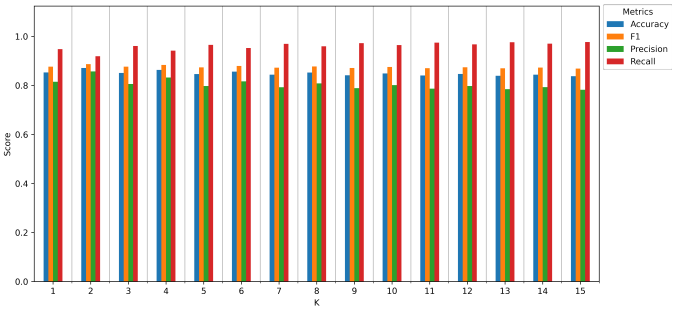


Fig. 2. Metrics Scores with respect to K value from 1 to 15

Fig. 3 presents the error rate (model accuracy = $100 - (error\_rate \times 100)$) of KNN with K ranging from 1 to 40. It can be seen from the figure that the error rate increases as K increases. In other words, the accuracy of the KNN models worsens as the K-value increases. The total runtime for this experiment was 153.4 minutes (approximately 2.5 hours).

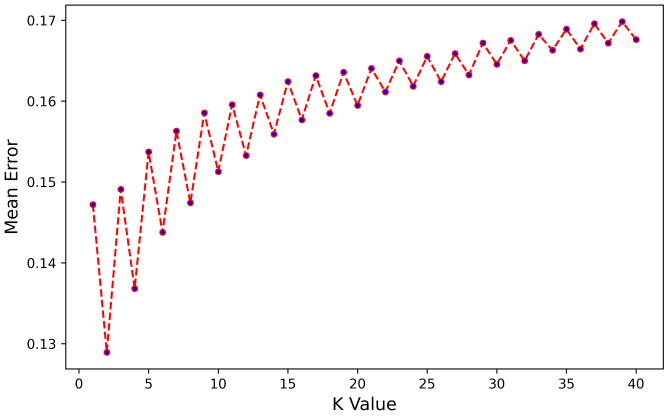


Fig. 3. Mean Error rate with respect to K value from 1 to 40

To confirm this assertion, we conducted the same experiment with K from 1 to 200, shown in Fig. (4). We can see from the figure that the model accuracy drops as the K increases. The total runtime for this experiment is 828 minutes (approximately 13.8 hours).

VI. CONCLUSION

This paper evaluated the performance of the KNN Machine Learning technique with the best performing K-value on the UNSW-NB15 dataset. The original UNSW-NB15 data set was first preprocessed with data cleaning, data transformation, and data reduction. The comprehensive preprocessing has undergone an iterative process, intending to improve and get better results. The KNN was then run extensively with a wide range of K values from 1 to 200. Based on our experiments for K-values from 1 to 200, we conclude that K equal to 2 performed the best with the highest model accuracy of 87.10%. Nonetheless, the results also indicate that increasing K-value results in worse performance in our case.

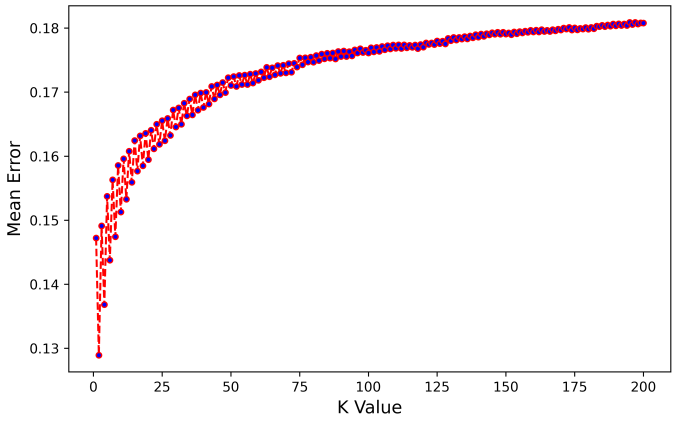


Fig. 4. Mean Error rate with respect to K value from 1 to 200

The results are meaningful due to the fact that there is a shortage of research on the KNN on our chosen dataset. Hence, future studies should address how the KNN model can be improved or developed to understand the implication of these results for our dataset. In addition, our research will be valuable for other practitioners who wish to develop the algorithm further, implying how the K-value affects the accuracy.

REFERENCES

[1] V. K. B. P, S. DV, S. Hegde, C. Pankaj, S. Suresh, and P. Ramakrishna, "Temporal based Network Packet Anomaly Detection using Machine Learning," in *2021 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, Jul. 2021, pp. 1–6.

[2] J. Long, F. Fang, and H. Luo, "A Survey of Machine Learning-based IoT Intrusion Detection Techniques," in *2021 IEEE 6th International Conference on Smart Cloud (SmartCloud)*, Nov. 2021, pp. 7–12.

[3] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network Anomaly Detection: Methods, Systems and Tools," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 303–336, 2014.

[4] B. Wang, C. Wang, W. Huang, Y. Song, and X. Qin, "A Survey and Taxonomy on Task Offloading for Edge-Cloud Computing," *IEEE Access*, vol. 8, pp. 186 080–186 101, 2020.

[5] G. Guo, "A Machine Learning Framework for Intrusion Detection System in IoT Networks Using an Ensemble Feature Selection Method," in *2021 IEEE 12th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, Oct. 2021, pp. 0593–0599.

[6] N. Moustafa, M. Keshky, E. Debiez, and H. Janicke, "Federated TON_iot Windows Datasets for Evaluating AI-Based Security Applications," in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. Guangzhou, China: IEEE, Dec. 2020, pp. 848–855.

[7] P. Jha and A. Sharma, "Framework to Analyze Malicious Behaviour in Cloud Environment using Machine Learning Techniques," in *2021 International Conference on Computer Communication and Informatics (ICCCI)*, Jan. 2021, pp. 1–12.

[8] U. Ahmad, H. Asim, M. T. Hassan, and S. Naseer, "Analysis of classification techniques for intrusion detection," in *2019 International Conference on Innovative Computing (ICIC)*, 2019, pp. 1–6.

[9] D. Jing and H.-B. Chen, "Svm based network intrusion detection for the unsw-nb15 dataset," in *2019 IEEE 13th International Conference on ASIC (ASICON)*, 2019, pp. 1–4.

[10] H. BENADDI, K. IBRAHIMI, and A. BENSLIMANE, "Improving the intrusion detection system for nsl-kdd dataset based on pca-fuzzy clustering-knn," in *2018 6th International Conference on Wireless Networks and Mobile Communications (WINCOM)*, 2018, pp. 1–6.

[11] S. Jaiswal, K. Saxena, A. Mishra, and S. K. Sahu, "A knn-aco approach for intrusion detection using kddcup'99 dataset," in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2016, pp. 628–633.

[12] L. Zhiqiang, G. Mohi-Ud-Din, L. Bing, L. Jianchao, Z. Ye, and L. Zhijun, "Modeling Network Intrusion Detection System Using Feed-Forward Neural Network Using UNSW-NB15 Dataset," in *2019 IEEE 7th International Conference on Smart Energy Grid Engineering (SEGE)*, Aug. 2019, pp. 299–303.

[13] M. Hammad, W. El-medany, and Y. Ismail, "Intrusion Detection System using Feature Selection With Clustering and Classification Machine Learning Algorithms on the UNSW-NB15 dataset," in *2020 International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT)*, Dec. 2020, pp. 1–6.

[14] M. Zeeshan, Q. Riaz, M. A. Bilal, M. K. Shahzad, H. Jabeen, S. A. Haider, and A. Rahim, "Protocol Based Deep Intrusion Detection for DoS and DDoS attacks using UNSW-NB15 and Bot-IoT data-sets," *IEEE Access*, pp. 1–1, 2021.

[15] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*, Nov. 2015, pp. 1–6.

[16] P. Wang, Y. Zhang, and W. Jiang, "Application of K-Nearest Neighbor (KNN) Algorithm for Human Action Recognition," in *2021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, vol. 4, Jun. 2021, pp. 492–496.

[17] Okfalisa, I. Gazalba, Mustakim, and N. G. I. Reza, "Comparative analysis of k-nearest neighbor and modified k-nearest neighbor algorithm for data classification," in *2017 2nd International conferences on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*, Nov. 2017, pp. 294–298.

Bibliography

[1] D. Sujana, S. Hegde, C. Pankaj, S. Suresh, P. Ramakrishna, *et al.*, "Temporal based network packet anomaly detection using machine learning," in *2021 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, IEEE, 2021, pp. 1–6.

[2] S. Naseer, Y. Saleem, S. Khalid, *et al.*, "Enhanced network anomaly detection based on deep neural networks," *IEEE Access*, vol. 6, pp. 48 231–48 246, 2018. DOI: 10.1109/ACCESS.2018.2863036.

[3] E. Anthi, L. Williams, M. Słowińska, G. Theodorakopoulos, and P. Burnap, "A supervised intrusion detection system for smart home iot devices," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 9042–9053, 2019. DOI: 10.1109/JIOT.2019.2926365.

[4] Z. K. Maseer, R. Yusof, N. Bahaman, S. A. Mostafa, and C. F. M. Foozy, "Benchmarking of machine learning for anomaly based intrusion detection systems in the cicids2017 dataset," *IEEE Access*, vol. 9, pp. 22 351–22 370, 2021. DOI: 10.1109/ACCESS.2021.3056614.

[5] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: Methods, systems and tools," *Ieee communications surveys & tutorials*, vol. 16, no. 1, pp. 303–336, 2013.

[6] M. Tavallaee, N. Stakhanova, and A. A. Ghorbani, "Toward credible evaluation of anomaly-based intrusion-detection methods," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 40, no. 5, pp. 516–524, 2010.

[7] J. Long, F. Fang, and H. Luo, "A survey of machine learning-based iot intrusion detection techniques," in *2021 IEEE 6th International Conference on Smart Cloud (SmartCloud)*, IEEE, 2021, pp. 7–12.

[8] B. Wang, C. Wang, W. Huang, Y. Song, and X. Qin, "A survey and taxonomy on task offloading for edge-cloud computing," *IEEE Access*, vol. 8, pp. 186 080–186 101, 2020.

[9] B. Steenwinckel, "Adaptive anomaly detection and root cause analysis by fusing semantics and machine learning," in *The Semantic Web: ESWC 2018 Satellite Events*, A. Gangemi, A. L. Gentile, A. G. Nuzzolese, *et al.*, Eds., Cham: Springer International Publishing, 2018, pp. 272–282, ISBN: 978-3-319-98192-5.

[10] T. M. Mitchell *et al.*, "Machine learning," 1997.

[11] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.

[12] I. C. Education, *What is machine learning?* Jul. 2020. [Online]. Available: https://www.ibm.com/cloud/learn/machine-learning.

[13] A. Networks, *What is anomaly detection? definition amp; faqs*, Apr. 2022. [Online]. Available: https://www-stage.avinetworks.com/glossary/anomaly-detection/.

[14] W. Haider, J. Hu, J. Slay, B. Turnbull, and Y. Xie, "Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling," *Journal of Network and Computer Applications*, vol. 87, pp. 185–192, 2017, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2017.03.018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804517301273.

[15] S. McElwee, *How to decide on a dataset for detecting cyber-attacks*, Feb. 2022. [Online]. Available: https://towardsdatascience.com/how-to-decide-on-a-dataset-for-detecting-cyber-attacks-c92e4f78e7a7.

[16] S. Stolfo, W. Lee, A. Prodromidis, and P. Chan, "Cost-based modeling for fraud and intrusion detection: Results from the jam project," vol. 2, Feb. 2000, 130–144 vol.2, ISBN: 0-7695-0490-6. DOI: 10.1109/DISCEX.2000.821515.

[17] M. Nour and S. Jill, *The unsw-nb15 dataset*, Nov. 2015. [Online]. Available: https://research.unsw.edu.au/projects/unsw-nb15-dataset.

[18] N. Moustafa and J. Slay, "Unsw-nb15: A comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*, 2015, pp. 1–6. DOI: 10.1109/MilCIS.2015.7348942.

[19] S. Choudhary and N. Kesswani, "Analysis of kdd-cup'99, nsl-kdd and unsw-nb15 datasets using deep learning in iot," *Procedia Computer Science*, vol. 167, pp. 1561–1573, 2020, International Conference on Computational Intelligence and Data Science, ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2020.03.367. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050920308334.

[20] T. Janarthanan and S. Zargari, "Feature selection in unsw-nb15 and kddcup'99 datasets," in *2017 IEEE 26th International Symposium on Industrial Electronics (ISIE)*, 2017, pp. 1881–1886. DOI: 10.1109/ISIE.2017.8001537.

[21] S. M. Kasongo and Y. Sun, "Performance analysis of intrusion detection systems using a feature selection method on the unsw-nb15 dataset," *Journal of Big Data*, vol. 7, no. 1, 2020. DOI: 10.1186/s40537-020-00379-6.

[22] M. F. Suleiman and B. Issac, "Performance comparison of intrusion detection machine learning classifiers on benchmark and new datasets," *2018 28th International Conference on Computer Theory and Applications (ICCTA)*, 2018. DOI: 10.1109/iccta45985.2018.9499140.

[23] E. Tufan, C. Tezcan, and C. Acarturk, "Anomaly-based intrusion detection by machine learning: A case study on probing attacks to an institutional network," *IEEE Access*, vol. 9, pp. 50 078–50 092, Jan. 2021. DOI: 10.1109/ACCESS.2021.3068961.

[24] A. Husain, A. Salem, C. Jim, and G. Dimitoglou, "Development of an efficient network intrusion detection model using extreme gradient boosting (xgboost) on the unsw-nb15 dataset," in *2019 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, 2019, pp. 1–7. DOI: 10.1109/ISSPIT47144.2019.9001867.

[25] W. Zong, Y.-W. Chow, and W. Susilo, "A two-stage classifier approach for network intrusion detection: 14th international conference, ispec 2018, tokyo, japan, september 25-27, 2018, proceedings," in Jan. 2018, pp. 329–340, ISBN: 978-3-319-99806-0. DOI: 10.1007/978-3-319-99807-7_20.

[26] M. Al-Daweri, K. A. Zainol Ariffin, S. Abdullah, and M. Senan, "An analysis of the kdd99 and unsw-nb15 datasets for the intrusion detection system," *Symmetry*, vol. 12, p. 1666, Oct. 2020. DOI: 10.3390/sym12101666.

[27] V. Kumar, D. Sinha, A. Das, D. S. Pandey, and R. Goswami, "An integrated rule based intrusion detection system: Analysis on unsw-nb15 data set and the real time online dataset," *Cluster Computing*, vol. 23, Jun. 2020. DOI: 10.1007/s10586-019-03008-x.

[28] T. Janarthanan and S. Zargari, "Feature selection in unsw-nb15 and kddcup'99 datasets," *2017 IEEE 26th International Symposium on Industrial Electronics (ISIE)*, pp. 1881–1886, 2017.

[29] G. Kocher and D. G. Kumar Ahuja, "Analysis of machine learning algorithms with feature selection for intrusion detection using unsw-nb15 dataset," *International Journal of Network Security Its Applications*, vol. 13, pp. 21–31, Jan. 2021. DOI: 10.5121/ijnsa.2021.13102.

[30] M. Ahmad, Q. Riaz, M. Zeeshan, H. Tahir, S. A. Haider, and M. S. Khan, "Intrusion detection in internet of things using supervised machine learning based on application and transport layer features using unsw-nb15 data-set," *EURASIP Journal on Wireless Communications and Networking*, vol. 2021, no. 1, 2021. DOI: 10.1186/s13638-021-01893-8.

[31] N. Moustafa and J. Slay, "The evaluation of network anomaly detection systems: Statistical analysis of the unsw-nb15 data set and the comparison with the kdd99 data set," *Information Security Journal: A Global Perspective*, vol. 25, no. 1-3, pp. 18–31, 2016. DOI: 10.1080/19393555.2015.1125974.

[32] Y. Li, W. Xu, and Q. Ruan, "Research on the performance of machine learning algorithms for intrusion detection system," *Journal of Physics: Conference Series*, vol. 1693, no. 1, p. 012 109, Dec. 2020. DOI: 10.1088/1742-6596/1693/1/012109. [Online]. Available: https://doi.org/10.1088/1742-6596/1693/1/012109.

[33] M. K. Islam, P. Hridi, M. S. Hossain, and H. S. Narman, "Network anomaly detection using lightgbm: A gradient boosting classifier," *2020 30th International Telecommunication Networks and Applications Conference (ITNAC)*, 2020. DOI: 10.1109/itnac50341.2020.9315049.

[34] N. Elmrabit, F. Zhou, F. Li, and H. Zhou, "Evaluation of machine learning algorithms for anomaly detection," in *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, 2020, pp. 1–8. DOI: 10.1109/CyberSecurity49315.2020.9138871.

[35] M. Hammad, W. El-medany, and Y. Ismail, "Intrusion detection system using feature selection with clustering and classification machine learning algorithms on the unsw-nb15 dataset," in *2020 International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT)*, 2020, pp. 1–6. DOI: 10.1109/3ICT51146.2020.9312002.

[36] Y. Pacheco and W. Sun, "Adversarial machine learning: A comparative study on contemporary intrusion detection datasets," *Proceedings of the 7th International Conference on Information Systems Security and Privacy*, 2021. DOI: 10.5220/0010253501600171.

[37] V. Kanimozhi and P. Jacob, *Unsw-nb15 dataset feature selection and network intrusion detection.* [Online]. Available: https://www.researchgate.net/profile/Prem-Jacob/publication/333175638_ES2080017519/links/5cdef62d92851c4eabaa
ES2080017519.pdf.

[38] G. C. Amaizu, C. I. Nwakanma, J.-M. Lee, and D.-S. Kim, "Investigating network intrusion detection datasets using machine learning," in *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, 2020, pp. 1325–1328. DOI: 10.1109/ICTC49870.2020.9289329.

[39] M. Al-Zewairi, S. Almajali, and A. Awajan, "Experimental evaluation of a multi-layer feed-forward artificial neural network classifier for network intrusion detection system," in *2017 International Conference on New Trends in Computing Sciences (ICTCS)*, 2017, pp. 167–172. DOI: 10.1109/ICTCS.2017.29.

[40] J. Yan, D. Jin, C. W. Lee, and P. Liu, "A comparative study of off-line deep learning based network intrusion detection," in *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2018, pp. 299–304. DOI: 10.1109/ICUFN.2018.8436774.

[41] R. A. Khamis and A. Matrawy, "Evaluation of adversarial training on different types of neural networks in deep learning-based idss," in *2020 International Symposium on Networks, Computers and Communications (ISNCC)*, 2020, pp. 1–6. DOI: 10.1109/ISNCC49221.2020.9297344.

[42] J. Brownlee, *Why data preparation is so important in machine learning*, Jun. 2020. [Online]. Available: https://machinelearningmastery.com/data-preparation-is-important/.

[43] *Levels of measurement: Nominal, ordinal, interval, ratio.* [Online]. Available: https://www.scribbr.com/statistics/levels-of-measurement/.

[44] A. Zhang, *Data types from a machine learning perspective with examples*, Nov. 2021. [Online]. Available: https://towardsdatascience.com/data-types-from-a-machine-learning-perspective-with-examples-111ac679e8bc.

[45] S. Maddali, *How important is data in machine learning?* Aug. 2021. [Online]. Available: https://medium.com/nerd-for-tech/how-important-is-data-in-machine-learning.

[46] Formplus, *Categorical data: Definition*, Oct. 2019. [Online]. Available: https://www.formpl.us/blog/categorical-data.

[47] J. Hale, *7 data types: A better way to think about data types for machine learning*, Apr. 2021. [Online]. Available: https://towardsdatascience.com/7-data-types-a-better-way-to-think-about-data-types-for-machine-learning.

[48] J. Brownlee, *How to avoid overfitting in deep learning neural networks*, Dec. 2018. [Online]. Available: https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/.

[49] Deepcheck, *What is overfitting in machine learning*, Aug. 2021. [Online]. Available: https://deepchecks.com/glossary/overfitting-in-machine-learning/.

[50] J. Brownlee, *Overfitting and underfitting with machine learning algorithms*, Mar. 2016. [Online]. Available: https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/.

[51] Geeksforgeeks, *Ml: Underfitting and overfitting*, Oct. 2021. [Online]. Available: https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/.

[52] S. Raschka, *Bias$_v$ariance$_d$ecomp: Bias-variance decomposition for classification and regression losses*, 2014. [Online]. Available: http://rasbt.github.io/mlxtend/user_guide/evaluate/bias_variance_decomp/#example-1-bias-variance-decomposition-of-a-decision-tree-classifier.

[53] P. Baheti, *Overfitting vs underfitting in machine learning [differences]*, 2022. [Online]. Available: https://www.v7labs.com/blog/overfitting-vs-underfitting.

[54] Seldon, *Supervised vs unsupervised learning explained*, Oct. 2021. [Online]. Available: https://www.seldon.io/supervised-vs-unsupervised-learning-explained.

[55] P. Cunningham, M. Cord, and S. J. Delany, "Supervised learning," in *Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval*, M. Cord and P. Cunningham, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 21–49, ISBN: 978-3-540-75171-7. DOI: 10.1007/978-3-540-75171-7_2. [Online]. Available: https://doi.org/10.1007/978-3-540-75171-7_2.

[56] P. Baheti, *Supervised vs. unsupervised learning [differences amp; examples]*, 2020. [Online]. Available: https://www.v7labs.com/blog/supervised-vs-unsupervised-learning.

[57] J. Brownlee, *4 types of classification tasks in machine learning*, Apr. 2020. [Online]. Available: https://machinelearningmastery.com/types-of-classification-in-machine-learning/.

[58] J. Delua, *Supervised vs. unsupervised learning: What's the difference?* Mar. 2021. [Online]. Available: https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning.

[59] J. Brownlee, *Difference between classification and regression in machine learning*, May 2017. [Online]. Available: https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/.

[60] Z. Ghahramani, "Unsupervised learning," in *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures*, O. Bousquet, U. von Luxburg, and G. Rätsch, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 72–112, ISBN: 978-3-540-28650-9. DOI: 10.1007/978-3-540-28650-9_5. [Online]. Available: https://doi.org/10.1007/978-3-540-28650-9_5.

[61] M. E. Celebi and K. Aydin, *Unsupervised Learning Algorithms*, 1st. Springer Publishing Company, Incorporated, 2016, ISBN: 3319242091.

[62] *Sklearn.model*$_s$*election.gridsearchcv*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

[63] *Tune hyperparameters with gridsearchcv*, Jul. 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/06/tune-hyperparameters-with-gridsearchcv.

[64] S. Shah, *Cost function: Types of cost function machine learning*, Feb. 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/02/cost-function-is-no-rocket-science/.

[65] G. Seif, *Understanding the 3 most common loss functions for machine learning regression*, May 2019. [Online]. Available: https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression/.

[66] J. Brownlee, *A gentle introduction to cross-entropy for machine learning*, Oct. 2019. [Online]. Available: https://machinelearningmastery.com/cross-entropy-for-machine-learning/.

[67] S. Saxena, *Binary cross entropy/log loss for binary classification*, Mar. 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/03/binary-cross-entropy-log-loss-for-binary-classification/.

[68] C. Mc., *Machine learning fundamentals (i): Cost functions and gradient descent*, Nov. 2017. [Online]. Available: https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220.

[69] R. M, *The ascent of gradient descent*, Sep. 2019. [Online]. Available: https://blog.clairvoyantsoft.com/the-ascent-of-gradient-descent-23356390836f.

[70] D. Goglia, *Backpropagation for dummies*, Jul. 2021. [Online]. Available: https://medium.com/analytics-vidhya/backpropagation-for-dummies-e069410fa585.

[71] DeepAI, *Feed forward neural network*, May 2019. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network.

[72] P. Baheti, *12 types of neural networks activation functions: How to choose?* May 2022. [Online]. Available: https://www.v7labs.com/blog/neural-networks-activation-functions/.

[73] M. Nielsen, *Neural networks and deep learning - latexstudio*, Dec. 2019. [Online]. Available: https://static.latexstudio.net/article/2018/0912/neuralnetworksanddeeplearning.pdf.

[74] L. Singh, *Forward and backward propagation-understanding it to master the model training process*, Jul. 2021. [Online]. Available: https://medium.com/geekculture/forward-and-backward-propagation-understanding-it-to-master-the-model-training-process-3819727dc5c1.

[75] H. J. Kelley, "Gradient theory of optimal flight paths," *Ars Journal*, vol. 30, no. 10, pp. 947–954, 1960.

[76] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[77] D. Johnson, *Back propagation neural network: What is backpropagation algorithm in machine learning?* May 2022. [Online]. Available: https://www.guru99.com/backpropogation-neural-network.html.

[78] A. Goh, "Back-propagation neural networks for modeling complex systems," *Artificial Intelligence in Engineering*, vol. 9, no. 3, pp. 143–151, 1995, ISSN: 0954-1810. DOI: https://doi.org/10.1016/0954-1810(94)00011-S. [Online]. Available: https://www.sciencedirect.com/science/article/pii/095418109400011S.

[79] R. Aswini Priyanka, C. Ashwitha, R. Arun Chakravarthi, and R. Prakash, "Face recognition model using back propagation," *International Journal of Engineering amp; Technology*, vol. 7, no. 3.34, p. 237, 2018. DOI: 10.14419/ijet.v7i3.34.18973.

[80] K.-L. Du and M. N. Swamy, "Multilayer perceptrons: Architecture and error backpropagation," *Neural Networks and Statistical Learning*, pp. 83–126, 2013. DOI: 10.1007/978-1-4471-5571-3_4.

[81] C. Hansen, *Activation functions explained - gelu, selu, elu, relu and more*, Aug. 2019. [Online]. Available: https://mlfromscratch.com/activation-functions-explained/#subscribe.

[82] M. Patel, *Vanishing and exploding gradients in neural networks*, Jan. 2021. [Online]. Available: https://medium.datadriveninvestor.com/vanishing-and-exploding-gradients-in-neural-networks-bddd4504e59c.

[83] Keshav, *Hyperbolic tangent (tanh) activation function [with python code]*, 2018. [Online]. Available: https://vidyasheela.com/post/hyperbolic-tangent-tanh-activation-function-with-python-code.

[84] ——, *Relu activation function [with python code]*, 2018. [Online]. Available: https://vidyasheela.com/post/relu-activation-function-with-python-code.

[85] K. He, X. Zhang, S. Ren, and J. Sun, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, 2015. DOI: 10.48550/ARXIV.1502.01852. [Online]. Available: https://arxiv.org/abs/1502.01852.

[86] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, *Fast and accurate deep network learning by exponential linear units (elus)*, 2015. DOI: 10.48550/ARXIV.1511.07289. [Online]. Available: https://arxiv.org/abs/1511.07289.

[87] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," 2017. DOI: 10.48550/ARXIV.1706.02515. [Online]. Available: https://arxiv.org/abs/1706.02515.

[88] T. Böhm, *An introduction to selus and why you should start using them as your activation functions*, Aug. 2018. [Online]. Available: https://towardsdatascience.com/gentle-introduction-to-selus-b19943068cd9.

[89] D. Hendrycks and K. Gimpel, *Gaussian error linear units (gelus)*, 2016. DOI: 10.48550/ARXIV.1606.08415. [Online]. Available: https://arxiv.org/abs/1606.08415.

[90] A. Shrivastav, *Gaussian error linear unit (gelu)*, Dec. 2021. [Online]. Available: https://iq.opengenus.org/gaussian-error-linear-unit/.

[91] K. E. Koech, *Softmax activation function - how it actually works*, Sep. 2020. [Online]. Available: https://towardsdatascience.com/softmax-activation-function-how-it-actually-works-d292d335bd78.

[92] M. Debnath, *Binary step function*, Nov. 2021. [Online]. Available: https://iq.opengenus.org/binary-step-function/.

[93] J. Cramer, *The origins of logistic regression*, Jan. 2003. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=360300.

[94] N. Cristianini and E. Ricci, "Support vector machines," in *Encyclopedia of Algorithms*, M.-Y. Kao, Ed. Boston, MA: Springer US, 2008, pp. 928–932, ISBN: 978-0-387-30162-4. DOI: 10.1007/978-0-387-30162-4_415. [Online]. Available: https://doi.org/10.1007/978-0-387-30162-4_415.

[95] A. P. Gopi, R. N. S. Jyothi, V. L. Narayana, and K. S. Sandeep, "Classification of tweets data based on polarity using improved RBF kernel of SVM," *International Journal of Information Technology*, 2020. DOI: 10.1007/s41870-019-00409-4.

[96] H.-T. Lin and C.-J. Lin, "A study on sigmoid kernels for svm and the training of non-psd kernels by smo-type methods," *Neural Computation*, Jun. 2003.

[97] X. Wu, V. Kumar, J. R. Quinlan, *et al.*, "Top 10 algorithms in data mining," *Knowledge and information systems*, vol. 14, no. 1, pp. 1–37, 2008.

[98] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986. DOI: 10.1007/bf00116251.

[99] J. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

[100] K. A. Grajski, L. Breiman, G. V. Di Prisco, and W. J. Freeman, "Classification of eeg spatial patterns with a tree-structured methodology: Cart," *IEEE Transactions on Biomedical Engineering*, vol. BME-33, no. 12, pp. 1076–1086, 1986. DOI: 10.1109/TBME.1986.325684.

[101] G. V. Kass, "An exploratory technique for investigating large quantities of categorical data," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 29, no. 2, pp. 119–127, 1980, ISSN: 00359254, 14679876. [Online]. Available: http://www.jstor.org/stable/2986296 (visited on 06/04/2022).

[102] J. H. Friedman, "Multivariate Adaptive Regression Splines," *The Annals of Statistics*, vol. 19, no. 1, pp. 1–67, 1991. DOI: 10.1214/aos/1176347963. [Online]. Available: https://doi.org/10.1214/aos/1176347963.

[103] S. Dobilas, *Mars: Multivariate adaptive regression splines - how to improve on linear regression?* Feb. 2022. [Online]. Available: https://towardsdatascience.com/mars-multivariate-adaptive-regression-splines-how-to-improve-on-linear-regression-e1e7a63c5eae.

[104] L. Rokach and O. Maimon, *Data Mining With Decision Trees: Theory and Applications*, 2nd. USA: World Scientific Publishing Co., Inc., 2014, ISBN: 9789814590075.

[105] L. Breiman, *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. DOI: 10.1023/a:1010933404324.

[106] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, IEEE, vol. 1, 1995, pp. 278–282.

[107] *Random forests trademark of minitab, llc - registration number 3185828 - serial number 78642027 :: Justia trademarks*. [Online]. Available: https://trademarks.justia.com/786/42/random-78642027.html.

[108] A. Mucherino, P. J. Papajorgji, and P. M. Pardalos, "Chapter 4," in *Data Mining in Agriculture*. Springer New York, 2009, pp. 83–106.

[109] P. Wang, Y. Zhang, and W. Jiang, "Application of K-Nearest Neighbor (KNN) Algorithm for Human Action Recognition," in *2021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, vol. 4, Jun. 2021, pp. 492–496. DOI: 10.1109/IMCEC51613.2021.9482165.

[110] Okfalisa, I. Gazalba, Mustakim, and N. G. I. Reza, "Comparative analysis of k-nearest neighbor and modified k-nearest neighbor algorithm for data classification," in *2017 2nd International conferences on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*, Nov. 2017, pp. 294–298. DOI: 10.1109/ICITISEE.2017.8285514.

[111] G. Cavallanti, N. Cesa-Bianchi, and C. Gentile, "Tracking the best hyperplane with a simple budget perceptron," *Machine Learning*, vol. 69, no. 2-3, pp. 143–167, 2007. DOI: 10.1007/s10994-007-5003-0.

[112] S. Patwardhan, *Knn algorithm: What is knn algorithm: How does knn function*, Apr. 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/04/simple-understanding-and-implementation-of-knn-algorithm/.

[113] G. Van Rossum and F. L. Drake Jr, *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[114] S. O. |. |. 5, S. O'Grady, and N. *, *The redmonk programming language rankings: June 2021*, Aug. 2021. [Online]. Available: https://redmonk.com/sogrady/2021/08/05/language-rankings-6-21/.

[115] Coursera, *What is python used for? a beginner's guide*, May 2022. [Online]. Available: https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python.

[116] T. Kluyver, B. Ragan-Kelley, F. Pérez, *et al.*, "Jupyter notebooks – a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt, Eds., IOS Press, 2016, pp. 87–90.

[117] Databricks, *What is a jupyter notebook?* Feb. 2022. [Online]. Available: https://databricks.com/glossary/jupyter-notebook.

[118] github, *Github*, 2020. [Online]. Available: https://github.com/.

[119] Kinsta, *What is github? a beginner's introduction to github*, May 2021. [Online]. Available: https://kinsta.com/knowledgebase/what-is-github/.

[120] L. Lamport, *Latex: A document preparation system, adison*, 1986.

[121] T. Oliphant, *Guide to NumPy*. Jan. 2006.

[122] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

[123] M. Waskom, O. Botvinnik, D. O'Kane, *et al.*, *Mwaskom/seaborn: V0.8.1 (september 2017)*, version v0.8.1, Sep. 2017. DOI: 10.5281/zenodo.883859. [Online]. Available: https://doi.org/10.5281/zenodo.883859.

[124] I. Srivastava, *Difference between matplotlib vs seaborn*, Nov. 2021. [Online]. Available: https://www.geeksforgeeks.org/difference-between-matplotlib-vs-seaborn/.

[125] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[126] T. Techopedia, *What is scikit-learn? - definition from techopedia*, Aug. 2019. [Online]. Available: https://www.techopedia.com/definition/33860/scikit-learn.

[127] Simplilearn, *What is keras and why it so popular in 2021: Simplilearn*, Sep. 2021. [Online]. Available: https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-is-keras.

[128] N. Moustafa and J. Slay, "The evaluation of network anomaly detection systems: Statistical analysis of the unsw-nb15 data set and the comparison with the kdd99 data set," *Information Security Journal: A Global Perspective*, vol. 25, no. 1-3, pp. 18–31, 2016. DOI: 10.1080/19393555.2015.1125974. eprint: https://doi.org/10.1080/19393555.2015.1125974. [Online]. Available: https://doi.org/10.1080/19393555.2015.1125974.

[129] N. Hoque, M. H. Bhuyan, R. Baishya, D. Bhattacharyya, and J. Kalita, "Network attacks: Taxonomy, tools and systems," *Journal of Network and Computer Applications*, vol. 40, pp. 307–324, 2014, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2013.08.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804513001756.

[130] K. Goyal, *Data preprocessing in machine learning: 7 easy steps to follow*, Jul. 2021. [Online]. Available: https://www.upgrad.com/blog/data-preprocessing-in-machine-learning/.

[131] DeepChecks, *What is normalization in machine learning*, Aug. 2021. [Online]. Available: https://deepchecks.com/glossary/normalization-in-machine-learning/.

[132] Zach, *How to read a correlation matrix*, Jan. 2020. [Online]. Available: https://www.statology.org/how-to-read-a-correlation-matrix/.

[133] K. E. Rao and G. A. Rao, "Ensemble learning with recursive feature elimination integrated software effort estimation: A novel approach," *Evolutionary Intelligence*, vol. 14, no. 1, pp. 151–162, 2020. DOI: 10.1007/s12065-020-00360-5.

[134] A. Dubey, *Feature selection using random forest*, Dec. 2018. [Online]. Available: https://towardsdatascience.com/feature-selection-using-random-forest-26d7b747597f.

[135] D. Medved, *Deep learning applications for biomedical data and natural language processing*, Aug. 2018. [Online]. Available: https://portal.research.lu.se/en/publications/deep-learning-applications-for-biomedical-data-and-natural-langua.

[136] M. Sunasra, *Performance metrics for classification problems in machine learning*, Feb. 2019. [Online]. Available: https://medium.com/@MohammedS/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b.

[137] D. Jing and H.-B. Chen, "Svm based network intrusion detection for the unsw-nb15 dataset," in *2019 IEEE 13th International Conference on ASIC (ASICON)*, 2019, pp. 1–4. DOI: 10.1109/ASICON47005.2019.8983598.

[138] G. Kocher and G. Kumar, "Performance analysis of machine learning classifiers for intrusion detection using unsw-nb15 dataset," *Computer Science amp; Information Technology (CS amp; IT)*, 2020. DOI: 10.5121/csit.2020.102004.

[139] R. Pujari, *Network attack detection and classification using machine learning models based on unsw-nb15.* Nov. 2020. [Online]. Available: https://i-rakshitpujari.medium.com/network-attack-detection-and-classification-using-machine-learning-models-based-on-unsw-nb15-a645bba73987.

[140] Q. Gu, Y. Chang, X. Li, Z. Chang, and Z. Feng, "A novel f-svm based on foa for improving svm performance," *Expert Systems with Applications*, vol. 165, p. 113 713, 2021. DOI: 10.1016/j.eswa.2020.113713.

[141] A. M. Ahmed, A. Rizaner, and A. H. Ulusoy, "A novel decision tree classification based on post-pruning with bayes minimum risk," *PLOS ONE*, vol. 13, no. 4, 2018. DOI: 10.1371/journal.pone.0194168.

[142] A. Ramesh, M. Pavlov, G. Goh, *et al.*, *Zero-shot text-to-image generation*, Feb. 2021. [Online]. Available: https://arxiv.org/abs/2102.12092.

[143] J. Adamczyk, *K nearest neighbors computational complexity*, Sep. 2020. [Online]. Available: https://towardsdatascience.com/k-nearest-neighbors-computational-complexity-502d2c440d5.

[144] B. Wang, X. Gan, X. Liu, *et al.*, "A novel weighted knn algorithm based on rss similarity and position distance for wi-fi fingerprint positioning," *IEEE Access*, vol. 8, pp. 30 591–30 602, 2020. DOI: 10.1109/ACCESS.2020.2973212.

BIBLIOGRAPHY