# Explainable AI by combining Neural Networks and Decision Trees

by Sanyog Sanjay Chavhan, MSc

Submitted to The University of Nottingham
in September 2024
in partial fulfilment of the conditions for the award of the degree of
Master of Science in Data Science

I declare that this dissertation is all my own work, except as indicated in the text

# Contents

## List of Figures

## Abstract

Neural networks, while highly accurate in predictive tasks, often suffer from a lack of transparency, a challenge commonly termed as the "black box" issue. This dissertation introduces an innovative strategy aimed at enhancing the interpretability of neural networks by augmenting them with decision trees. The approach involves using synthetic classification datasets that feature a variety of complex geometrical shapes like linear, circular, elliptical, and square. These shapes serve as decision boundaries that model complex real-world classification scenarios. A single derived feature is created for each dataset to capture essential data patterns, enabling decision trees to effectively mirror the neural networks' decision-making processes.

This research applies neural networks, with the MLP being one example, to classify complex decision boundaries, and uses decision trees to replicate these boundaries, leveraging both the original features and an added derived feature for interpretability. The effectiveness and clarity of these decision trees are then evaluated against interpretable models such as Linear Support Vector Machines (SVM) and Logistic Regression with polynomial features. The findings indicate that integrating the derived feature that captures the essence of the underlying data significantly enhances the decision trees' capability to mimic the neural networks' decisions while preserving accuracy.

This investigation enriches the field of Explainable Artificial Intelligence (XAI) by demonstrating a hybrid model that effectively balances accuracy with interpretability. The results advocate that merging neural networks with decision trees, complemented by strategic features, offers a robust solution for crafting AI systems that are both effective and interpretable. This methodology holds particular value in areas where the transparency of AI decisions is critical.

Keywords: Explainable Artificial Intelligence (XAI), Neural Networks, Decision Trees, Model Interpretability, Surrogate Models, AI systems, Feature Engineering, Black-Box Models, Interpretable Machine Learning

## Acknowledgements

I would like to extend my deepest appreciation to my supervisor, Professor Andrew Parkes, for his expert guidance, insightful critiques, and constant encouragement throughout the duration of my research. His invaluable advice and steadfast support were crucial to the success of this research. I am also profoundly grateful to my parents, whose unwavering support and boundless love provided me with the strength and motivation needed to pursue my academic goals. Their belief in my abilities has been a constant source of inspiration. Additionally, I owe a special thank you to my friends, whose companionship and understanding helped me maintain my focus and composure through the challenging times of this dissertation. Their support has been a pillar of my resilience.

Dedicated to my parents and my partner, for their endless support, love, and encouragement throughout this journey.

# Chapter 1: Introduction

## 1.1 Background and Context

The development and deployment of artificial intelligence (AI) technologies have had a huge impact on various sectors by bringing about advanced levels of data processing, forecasting, and making decisions. Although this technology has enormous potential in key areas including healthcare, finance, and law enforcement, its advancement has been hampered by the opaqueness of AI systems and lack of transparency of its industrial use. These models, while powerful, are often referred to as "black boxes" due to their complex internal structures and opaque decision-making processes, making it challenging for users to understand how predictions are made (Heaton, 2018). To address this challenge, the field of Explainable AI (XAI) has emerged, focusing on developing methods that enhance the interpretability and transparency of AI models without compromising their predictive performance (Doshi-Velez and Kim, 2017).

Neural networks, particularly Multi-Layer Perceptrons (MLPs), are highly effective in capturing complex, non-linear relationships within data, yet they lack intrinsic interpretability (Zhang et al., 2020). On the other hand, decision trees are inherently transparent and provide rule-based explanations, making them ideal for contexts where understanding the model's decision process is crucial. This research explores a novel approach that integrates the strengths of both neural networks and decision trees, aiming to bridge the gap between high performance and interpretability.

## 1.2 Problem Statement

While neural networks excel in modelling complex decision boundaries, their lack of transparency creates challenges in high-stakes applications where understanding the decision-making process is important. Existing XAI methods, such as Local Interpretable Model-agnostic Explanations (LIME) and Shapley Additive Explanations (SHAP), provide some insights into individual predictions but often fail to explain the model as a whole. This study addresses the gap by investigating how decision trees can be employed as surrogate models to approximate the decision-making process of neural networks, thereby enhancing interpretability without sacrificing accuracy.

## 1.3 Research Objectives

The primary objectives of this research are:

1. To develop a method that augments neural networks with decision trees to improve interpretability.
2. To derive and validate a simplified feature that aids decision trees in mimicking the decision boundaries created by neural networks.
3. To evaluate the performance of this hybrid model across various types of decision boundaries (linear, oblique, circular, elliptical, and square-shaped).

## 1.4 Research Questions

Based on the objectives, this study seeks to answer the following research questions:

1. How can decision trees be effectively used to mimic the decision boundaries of neural networks while maintaining high accuracy?
2. What types of derived features can facilitate decision trees in approximating the decision-making process of neural networks?
3. Does the hybrid model demonstrate superior interpretability compared to neural networks alone across different types of decision boundaries?

## 1.5 Significance of the study

This study contributes to the growing field of XAI by proposing a practical approach to making neural networks more interpretable. The ability to augment neural networks with decision trees opens new possibilities for their use in critical applications where trust, transparency, and accountability are paramount. By enhancing the interpretability of these models, this research addresses a crucial barrier to the wider adoption of AI technologies in sensitive areas such as healthcare, finance, and law enforcement, where understanding AI decisions is vital for compliance, trust, and effective decision-making (Rudin, 2019) (Gunning *et al.*, 2019).

## 1.6 Structure of the dissertation

This dissertation is structured as follows:

- Chapter 2: Theoretical Background and Literature Review – Provides an overview of relevant literature on XAI, neural networks, and decision trees, and discusses existing methods for improving model interpretability.
- Chapter 3: Methodology – Describes the research design, data generation processes, model training procedures, and evaluation metrics used in this study.
- Chapter 4: Experiments and Results – Provides Experimental Setup, presents the findings from the experiments, including the performance of neural networks, decision trees, and the hybrid model.
- Chapter 5: Discussion – Analyses the results, discusses the implications of the findings, and compares the proposed method with existing interpretability techniques.
- Chapter 6: Conclusion – Summarizes the key findings, suggests future research directions, and provides concluding remarks.

# Chapter 2: Theoretical Background and Literature Review

This chapter provides a thorough examination of the current body of literature concerning Explainable AI (XAI), with a specific emphasis on the incorporation of Neural Networks and Decision Trees. The goal is to provide a comprehensive coverage of the principles, approaches, and current research in Explainable Artificial Intelligence (XAI). The following sections will address fundamental topics such the concept and significance of Explainable Artificial Intelligence (XAI), the strengths and limitations of Neural Networks, the interpretability of Decision Trees, and the integration of these two methodologies to enhance interpretability. Additionally, existing methods for model interpretability will be reviewed, and a gap in the current research will be identified.

## 2.1 *Overview of Explainable AI (XAI)*

Explainable Artificial Intelligence (XAI) refers to techniques that make the output of AI models more interpretable and understandable to humans. The increasing utilisation of AI in critical applications like healthcare, banking, and law enforcement has underlined the necessity for transparency in AI decision-making (Saraswat *et al.*, 2022). Recent advancements in Explainable Artificial Intelligence (XAI) have mostly concentrated on improving the interpretability of black-box models such as Neural Networks. Despite their high accuracy, these models lack inherent transparency. Although techniques such as Local Interpretable Model-agnostic Explanations (LIME)(Ribeiro, Singh and Guestrin, 2016), Shapley Addictive Explanations (SHAP)(Štrumbelj and Kononenko, 2014), and Gradient Weighted Class Activation Map (Grad-CAM)(Selvaraju *et al.*, 2017) are frequently employed to explain individual predictions, however they generally do not provide insights into the model's whole decision-making process (Minh *et al.*, 2022).

Despite significant advancements, numerous hurdles persist in the development of fully transparent AI. An essential challenge is to achieve a balance between the trade-off between model performance and interpretability (Minh *et al.*, 2022). For instance, simpler models like Decision Trees provide intrinsic interpretability but may not achieve the same level of accuracy as compared to more complex models like Neural Networks. Therefore, it is imperative to investigate techniques that integrate the advantages of disparate AI models to achieve both high accuracy and transparency in decision-making (Blanco-Justicia *et al.*, 2020).

## 2.2 *Neural Networks: Architecture, Strengths and Limitations*

Inspired by the human brain, Neural Networks are a type of machine learning models featuring layers of linked neurons. These neurons are interlinked with each other in such a way that they process and transform the input data into an output. Neural Networks excel in tasks that involve analysing huge datasets and identifying complicated patterns, which makes them ideal for applications like image recognition, natural language processing and financial forecasting (Ismail Fawaz *et al.*, 2019). This section delves into the core architecture of neural networks pertinent to this dissertation, with a special emphasis on Multi-Layer Perceptrons (MLPs) which have been used extensively throughout this research.

### 2.2.1 *Architecture of a Neural Network:*

Neural Networks are composed of multiple layers of interconnected neurons. The fundamental component of a neural network is the neuron, which takes inputs and generates an output through a mathematical function also known as an activation function (Chakraborty, 2024). Below is a diagram that illustrates the biological structure of a neuron along with the structure of a simple neural network:

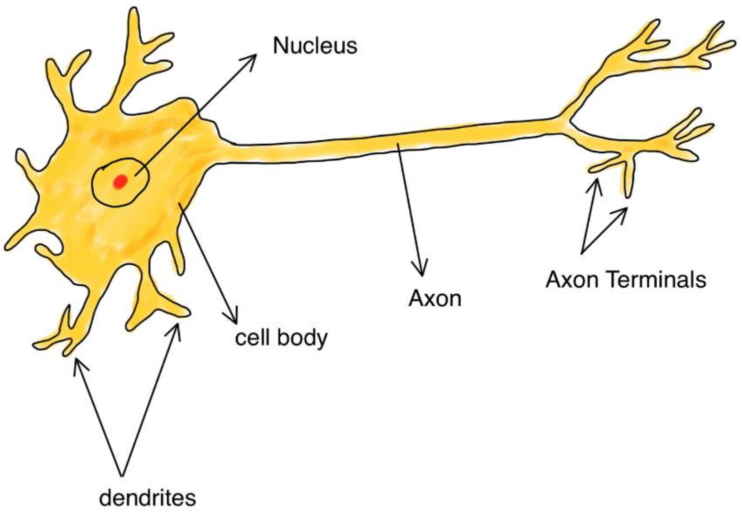

**Figure 2.1:** Biological Structure of a Neuron (Inspired by Richard Nagyfi, "The Differences Between Artificial and Biological Neural Networks," *Towards Data Science*, October 20, 2021. Available at: https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7)

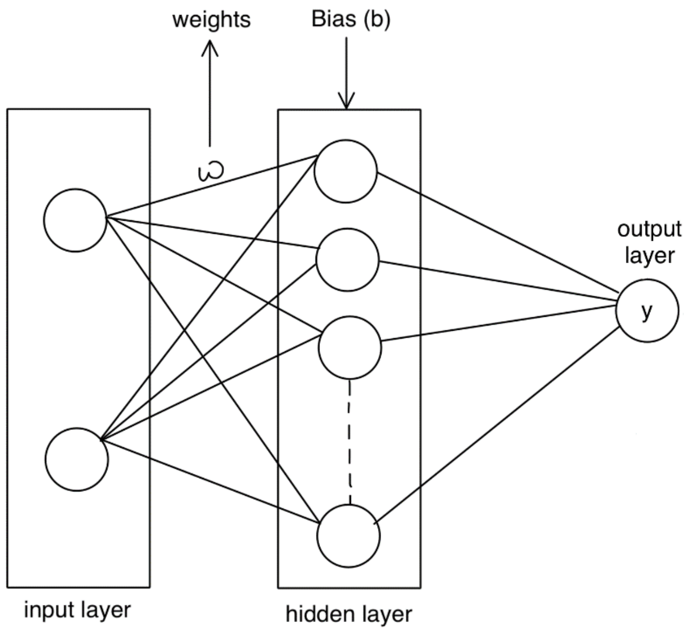

**Figure 2.2:** Structure of a simple Neural Network

The variety of neural network architectures cater to different task requirements:

1. **Perceptron:** As the simplest neural network model, the perceptron forms the foundation for more advanced networks. It includes a single layer where an input node is linked directly to an output node. This model utilises a weighted sum of the inputs, processes it through an activation function like a step function, and produces a binary outcome. Although effective for binary classification tasks, its application is restricted to linearly separable challenges, and it struggles with more complex, non-linear issues.

2. **Feed-Forward Neural Networks:** Expanding on the perceptron, feed-forward or Fully Connected Networks consist of multiple layers including an input layer, several hidden layers, and an output layer. Each neuron in one layer connects to all neurons in the subsequent layer, creating a densely interconnected structure. These networks adjust their neuron weights during training to reduce prediction errors. Known as "feed-forward" due to the unidirectional flow of data from input to output, these networks can, however, be prone to overfitting, especially with numerous layers or nodes (Aytekin, 2022).

3. **Multi-Layer Perceptron (MLP):** An MLP is a type of feed-forward network featuring multiple layers of nodes, typically with an input layer, several hidden layers, and an output layer (Frosst and Hinton, 2017). Each perceptron in a layer connects to every perceptron in the following layer. MLPs can model complex, non-linear functions using activation functions such as sigmoid, tanh, or ReLU, which introduce non-linearities. The hidden layers enable the network to learn more complex features and patterns within the data.

An MLP's architecture allows it to handle more complex relationships than a single-layer perceptron, such as approximating any continuous function, thus enhancing its utility for tasks like image and speech recognition or natural language processing.

### *2.2.2 Strengths of a Neural Network:*

1. **High Predictive Accuracy:** Neural Networks stand out in classification and regression due to their ability to understand complex non-linear data relationships (Chakraborty, 2024).
2. **Scalability:** Neural Networks are adept at processing huge datasets and can be effectively scaled-up for various applications (Ismail Fawaz *et al.*, 2019).

### *2.2.3 Limitations of a Neural Network:*

1. **Lack of Interpretability:** Neural Networks despite their high predictive accuracy are often referred to as "black box" models because their decision-making process is not transparent (Duede, 2023).
2. **Overfitting:** Neural Networks may adapt too closely to training data, particularly if they overly complex or trained on limited data (Aytekin, 2022).
3. **Computational Complexity:** Significant computational resources are required to train Neural Networks, which might possess a barrier in limited resource settings (Chakraborty, 2024).

This dissertation leverages the capabilities of MLPs to achieve explainable AI by integrating them with Decision Trees. The decision to utilise MLPs is based on their strengths and their appropriateness for the classification tasks involved in this research:

1. **Complex Relationship Learning:** MLPs are equipped with capturing complex, non-linear relationships within datasets. Given the data having diverse decision boundaries which we will be studying in this research, which includes linear, circular, elliptical, and square, MLPs are ideally suited for accurately classifying the features based on the type of decision boundary.
2. **Universal Approximation Feature:** With one or more hidden layers and non-linear activation functions, MLPs theoretically can approximate any continuous function with high accuracy, crucial for modelling complex decision boundaries in this research (Cybenko and Cybenkot, 1989).

This initial review of Neural Networks, particularly emphasising the function of Multi-Layer Perceptron and their ability to model complex data structures, paves the way for the next section, where we will delve into Decision Trees. These models are inherently transparent and interpretable, qualities that enhance the functionality of Neural Networks in building a more transparent AI framework.

## *2.3 Decision Trees: Understanding, Interpretability and Explainability*

Decision Trees are a form of supervised learning algorithm renowned for their natural interpretability and explainability. Unlike many other machine learning models that are considered "black boxes," Decision Trees offer a clear and transparent decision-making process, which is invaluable in scenarios where it's important to understand the basis of predictions (Blanco-Justicia *et al.*, 2020). This section offers a comprehensive review of Decision Trees, detailing their structure, functionality, and their importance in improving model interpretability for this dissertation.

### *2.3.1 Understanding Decision Trees*

A Decision Tree is a flowchart-like model in which each internal node represents a "test" or decision on an attribute (feature), each branch represents the outcome of that test, and each leaf node represents a class label (decision) (Aytekin, 2022). The path from the root node to a leaf node represents a classification rule (Ferigo, Custode and Iacca, 2023). Decision Trees are suited for both classification and regression tasks.

A typical structure of a Decision Tree has been illustrated in the below figure:

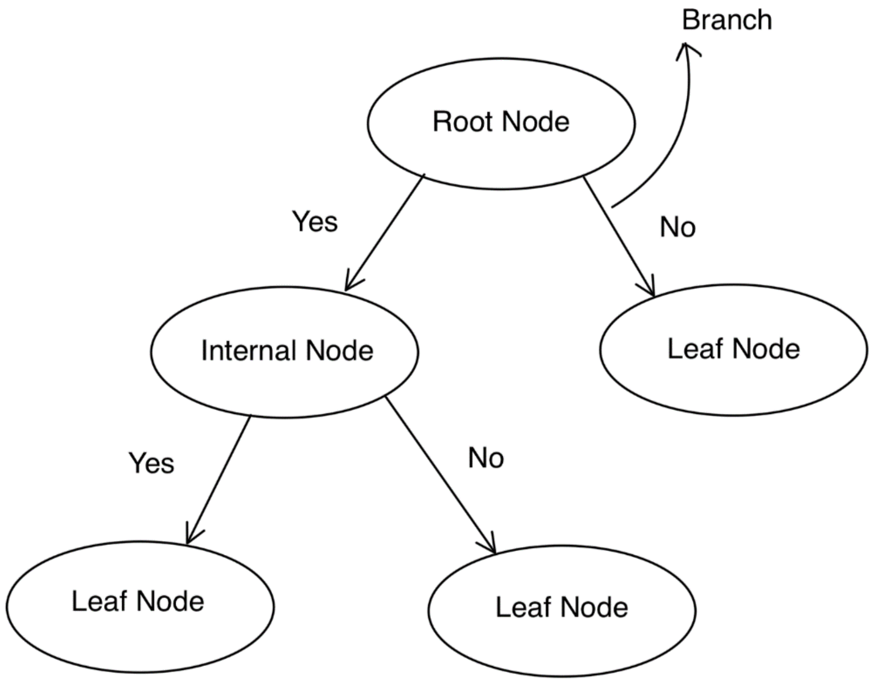

**Figure 2.3:** Basic Structure of a Decision Tree

**Root Node:** The root node represents the entire dataset and contains the initial feature on which the first split is made. This feature is selected based on a criterion that optimises the decision-making process, such as Information Gain or Gini Index.

**Internal Nodes:** Internal nodes represent tests on various features. Each node splits the data into two or more subsets based on the test condition, such as "Is feature Y greater than a certain value?".

**Branches:** Each branch represents the outcome of a test and leads to the next node or a leaf.

**Leaf Nodes:** Leaf nodes are the terminal nodes that provide the final decision or prediction. They represent the outcome for the instances that have followed the specific path from the root through the internal nodes.

Types of Decision Trees:

1. **Classification Trees:** Classification trees are used when the target variable is categorical. The tree is constructed to predict the class label based on input features. The goal is to find splits that maximise the purity of the leaf nodes (i.e., nodes containing only instances from a single class).
2. **Regression Trees:** Regression trees are used when the target variable is continuous. The tree aims to predict numerical values by minimising the variance within each leaf.

### 2.3.2 Interpretability and Explainability

In this dissertation, Decision Trees are utilised to improve the interpretability of Neural Networks. By taking advantage of the inherent transparency of Decision Trees, this research aims to unravel the “black box” nature of Neural Networks, thus fostering an AI system that is both accurate and interpretable.

**Rule-Based Interpretability:** Each path from a root node to a leaf node in a decision tree represents a decision rule that can be further understood and validated. By extracting these rules, one can get insights into the model’s decision-making process which is an important aspect for applications where Explainable AI is required (Frosst and Hinton, 2017).

**Surrogate Models for Explanations:** Decision Trees can act as surrogate models to approximate the behaviour of more complex models like Neural Networks. One can build an interpretable model that behaves in the same manner as neural networks by training a Decision Tree based on the neural network’s predictions (Craven and Shavlik, 1997) (Gupta, Park and Lam, 1999). This is very helpful in explaining the decision to stakeholders who do not have to interact with the ocean of the presented complex system of a neural network.

The next section will explore the integration of Neural Networks and Decision Trees, demonstrating how the predictive capabilities of the neural network can be combined with the interpretability of decision trees to create a more explainable AI model that focuses on transparency issues.

## 2.4 Combining Neural Networks and Decision Trees for Improved Interpretability

Merging Neural Networks and Decision Trees constitutes an appealing strategy for developing AI systems that retain a high predictive capability along with interpretability. With this strategy, both models complement each other: Neural Networks bring the ability to learn more complex patterns in data, while the inherent transparency of Decision Trees brings clear rule-based explanations (Craven and Shavlik, 1997). This section delves into the aspects considered in the integration, and relevant literature that has explored similar approaches.

### 2.4.1 Aspects considered in the integration

1. **Balancing Accuracy and Interpretability:** A fundamental consideration is finding the right balance between the complexity of the Neural Network and the interpretability of the Decision Tree. The goal is to develop a system where the Decision Tree can provide meaningful insights into the decision-making process of the Neural Network without significantly compromising the model's accuracy.
2. **Mitigating the Black-Box Problem:** One of the primary motivations behind this integration is to mitigate the "black-box" problem associated with Neural Networks. By incorporating Decision Trees, it becomes possible to open the black box, providing a clearer view of how predictions are made, which is crucial for model validation, auditing, and ensuring compliance with ethical guidelines.

Several studies have examined the integration of Neural Networks and Decision Trees to enhance the interpretability of models while keeping or even improving the accuracy. This section reviews four key papers that illustrate various approaches and use cases for such integration.

(Craven and Shavlik, 1997) introduced TREPAN algorithm, which is created to facilitate rule extraction from the trained Neural Networks. As its name suggests, it is an iterative query and sampling approach called TREPAN that aims at acquiring the information related to the decision borders of the neural network by reconstructing its parts. This method brings the complex decisions of the network into simplified rules for better understanding without compromising the accuracy of the original model's decisions. The application of this method is especially valuable in fields such as healthcare and finance, where understanding model predictions is critical for compliance and building trust.

(Gupta, Park and Lam, 1999) proposed the GLARE algorithm that pairs Decision Tree and Neural Networks to enhance the interpretability. GLARE offers something new in the case of a trained feedforward neural network, it translates the weights into understandable rules. Considering the weight magnitudes of the network, the algorithm formulates decision rules in such a way that the importance of how each input influences the decision system is revealed. This approach not only simplifies understanding but also closely mirrors the original network's behaviours. The method is especially effective in the domains of classification problems in particular image understanding or data classification where clear and precise models are required for effective decision-making and diagnostics.

(Krishnan, Sivakumar and Bhattacharya, 2002) developed DecText, a method for increasing the interpretability of output from trained Neural Networks by extracting decision trees from them, which are of structure like C4.5 model. DecText builds a decision tree which reuses the decision outputs as input through rules to recreate the decisions of the neural network. The goal is to formulate decision rules that are quite sure of neural networks output using inferences and better understanding. This approach is particularly suitable in cases of large databases as it helps to ensure that the complexity of the neural networks does not affect their interpretability. By deploying decision trees, the research tackles neural network's "black box" issue, enhancing their applicability in sectors like healthcare and finance where transparency is crucial.

Considering such integration approaches and their applications, this dissertation takes on a similar hybrid approach to leverage both Neural Networks and Decision Trees. This approach seeks to solve the drawbacks of the "black box" nature of neural networks while maintaining the predictive power of the neural networks and further promote the development of trustworthy and transparent AI systems in sensitive domains.

## 2.5 Existing Techniques for Model Interpretability

Several other existing approaches are aimed at improving AI models interpretability including approaches that seek to explain the reasoning behind complex models like Neural Networks. This section discusses the key techniques devised for the purpose of making the model interpretable by focusing on key studies that were instrumental in the development of such strategies.

(Ribeiro, Singh and Guestrin, 2016) introduced LIME (Local Interpretable Model-agnostic Explanations) in their influential paper "Why should I trust you? Explaining the predictions of any classifier". It is the purpose of LIME to provide interpretations for the output of any such black box classifier by fitting a simple interpretable model to a local region of the prediction space. It works by disturbing the input data and looking at the output, fitting a simple linear model which is valid in the neighbourhood of a certain prediction and describes how the complex model behaves in that region. This is particularly useful for models where the global interpretation of the system, for example, a Neural Network, is not possible but local interpretation on how individual predictions are made can still be insightful. LIME also supports cross feature explanations since it adheres to the model agnostic principle with application to healthcare, finance, customer behaviour and many other fields where understanding specific predictions is critical.

In the study by (Lundberg, Allen and Lee, 2017), which they entitled "A Unified Approach to Interpreting Model Predictions," the authors introduced SHAP, which stands for SHapley Additive exPlanations and aims to interpret a complex machine learning model. SHAP operates on the principles of cooperative game theory and uses Shapley values to quantify the importance of each feature in contributing to a specific prediction. To address this problem, the authors propose a generic approach ensuring consistency and fairness of feature changes across multiple models. Furthermore, in contrast to LIME that concentrates on local approximations, SHAP is a global rather than a local interpretation and aggregates Shapley values over all instances to get the behaviour of the model in its entirety. This technique is particularly valued for its theoretical foundations, which ensure that the feature attributions are consistent and additively explain the model output, making it suitable for high-stakes domains where fairness and transparency are essential.

(Friedman and Popescu, 2008) created a novel methodology known as the RuleFit algorithm, whereby linear models and decision-making rules are used in a bid to increase the models' interpretability. In "Predictive Learning via Rule Ensembles," they propose a model which extends decision tree rules to sparse linear models; hence the final model represents not only linear but also interaction between features and non-linearities. For RuleFit, rules are induced from the data by constructing a tree, and the so induced rules features are incorporated into a linear model which uses regularization. It is this combination of features which gives the model ability to be interpretable yet capture complex and nonlinear relationships within the data. The RuleFit approach is especially efficient in the situations when the modeler must provide understandable rules or explanations of the decisions made by the model to domain experts such as credit scoring, clinical decision support and risk assessment models.

(Lundberg, Erion and Lee, 2018) added an interesting implementation for tree-based models into the SHAP framework called the TreeSHAP, described in the paper titled 'Consistent Individualized Feature Attribution for Tree Ensembles'. TreeSHAP is an algorithm that can be used to compute Shapley values for tree models, including Decision Tree, Random Forest, and Gradient Boosting Tree. This approach employs a data structure –decision tree - to accelerate the computation of variable effects, hence is less computationally intensive than the conventional SHAP method. Most of the desirable properties of the original SHAP method e.g. consistency and local accuracy is maintained in TreeSHAP; considering also that this version was developed primarily for tree models where scalability was required. This makes it particularly useful in applications involving large datasets or complex ensembles, such as environmental modelling, fraud detection, and financial forecasting.

In their noteworthy publication titled "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization," (Selvaraju *et al.*, 2017)introduced a method called Grad-CAM which stands for Gradient-weighted Class Activation mapping. Grad-CAM is a technique designed specifically for convolutional neural networks (CNNs) to provide visual explanations of model predictions. The method uses the gradients of a target output class flowing into the final convolutional layer to produce a heatmap that highlights the regions of the input image most relevant to the model's decision (Cortés-Ferre *et al.*, 2023). With this technique, one can see which region of the image contributed towards a specific classification which is highly beneficial in fields such as medical imaging, autonomous driving and surveillance where it is important to know what the model is focusing on. Grad-CAM has gained a significant amount of attention as an explanation method in CNNs as it achieves and simplifies the generation of visual explanations with no modification to the model architecture.

In addition, these five techniques depict the nuances of how various approaches are taken to make AI models Explainable from local approaches such as LIME to the global methods such as SHAP and TreeSHAP to rule-based approaches such as RuleFit and finally visualisation methods such as Grad-CAM. Each of these various approaches has its own advantages and seeks to solve different parts of the problem of model interpretability, which is beneficial for understanding complex models in diverse applications.

## 2.6 Gap in Current Research

For all the efforts made in developing and implementing tools to enhance model interpretability, explanations of how even complex models like Neural Networks arrive at decisions still leave much to be desired. Existing interpretable artificial intelligence (XAI) techniques, including LIME, SHAP, Grad-CAM, tend to be focused on individual model outputs and the provision of related feature attribution scores or model's focused attention areas rather than the explanation on the performance of the model as a whole. However, these methods do not provide a comprehensive explanation of the overall model behaviour, particularly the underlying decision boundaries that neural networks learn to separate different classes of data. This limitation is crucial, especially in scenarios where stakeholders require a transparent understanding of how the model makes decisions across the entire dataset, not just for individual predictions.

This dissertation seeks to address this gap by proposing a new technique that integrates Neural Networks with Decision Trees that enhances interpretability without sacrificing accuracy. In classification tasks where a Neural Network classifies complex data with diverse decision boundaries, the goal is to combine the predictive power of Neural Networks with the inherent transparency of Decision Trees. By using the Neural Network's predictions and the input data, this approach seeks to derive a mathematical representation or rule that approximates the decision boundary formed by the Neural Network to classify the data. Unlike methods that generate multiple, complex rules such as RuleFit, this approach focuses on producing a simplified mathematical representation that reflects the underlying data's nature, thereby providing a clearer understanding of the Neural Network's "black box" characteristics. This integration addresses the critical need for a method that not only interprets individual predictions but also sheds light on the global behaviour of Neural Networks, making AI models more transparent and trustworthy in real-world applications.

## 2.7 Summary

This literature review has focused on important research and techniques in Explainable AI, with an emphasis on integrating Neural Networks with Decision Trees to improve interpretability. Although Neural Networks handle complex data effectively, their "black-box" nature complicates transparency efforts, a problem that Decision Trees help solve with their rule-based explanations. We have considered interpretability methods like LIME (Ribeiro, Singh and Guestrin, 2016), SHAP (Štrumbelj and Kononenko, 2014), RuleFit (Friedman and Popescu, 2008), TreeSHAP (Lundberg, Erion and Lee, 2018), and Grad-CAM (Selvaraju *et al.*, 2017), which provide insights into individual predictions but often fail to explain the overall model behaviour. A distinct gap persists in fully grasping the decision-making mechanisms of Neural Networks at a global level. This dissertation intends to bridge this gap by combining Neural Networks with Decision Trees, thereby simplifying the representation of decision boundaries through mathematical models, enhancing interpretability without losing accuracy, and contributing to the development of more transparent AI systems.

# Chapter 3: Methodology

## *3.1 Research Design*

This study's research design focuses on the core aim of augmenting the interpretability of neural networks by integrating them with decision trees. This method addresses the intrinsic challenges associated with neural networks, which are often perceived as "black box" models because their complex internal mechanisms conceal the logic behind their predictions. By utilising decision trees, known for their inherent transparency and rule-based explanations, this research intends to close the gap between high model performance and interpretability.

The research adopts an exploratory and experimental approach, by generating 2D synthetic classification datasets designed to represent various decision boundary scenarios like linear, circular, elliptical, and square shapes. This variety allows for an exhaustive examination of how different data geometries affect the interpretability of neural network models when coupled with decision trees.

**Data Generation and Preprocessing:** In this case, we generate 2D synthetic classification datasets having different geometries with each dataset representing a different decision boundary problem. Such testing ensures fundamental understanding of the structures through experimental means. Here the steps involve normalisation and transformation to prepare the data for model training in the correct form.

**Feature Engineering:** A specific feature, which is derived from each of the datasets is formulated to capture the essential characteristics of the underlying data. Although represented as a single value in this study, it could be generalised to a vector; for example, using a formula like $x^2 + y^2$ to characterise circular pattern in the data. This feature can be a representation of more complex geometric relationships, depending on the underlying data structure. By incorporating this feature along with the input data and the predictions from the neural network, the decision tree is provided with a comprehensive set of information. This enhanced input allows the decision tree to accurately mimic the neural network's classification logic without sacrificing accuracy and interpretability.

**Model Training:** In particular, we focus on the construction of two main models: a neural network (e.g., Multi-Layer Perceptron in this case) that creates a class label for a given set of features and a decision tree that aims to reproduce the predictions made by the neural network. The decision tree is trained using both the input features; for example, the input features could look like $(x, y)$, along with the predictions made by the neural network; for example, for each $(x, y)$ pair, the neural network will classify that point into either of the two classes, along with this a feature that provides the essence of the underlying data is also provided to the decision tree. This will effectively result in the decision tree imitating the neural network's classification logic, with the key difference being that the decision tree's outputs are transparent.

**Model Evaluation:** To quantify the performance of the two obtained models, we used classical classification metrics, including accuracy, precision, recall, and F1-score. In addition, also the evaluation of the decision tree was performed based on how accurately the obtained mathematical rule can classify the data just like the neural network.

**Comparative Analysis:** An in-depth comparative analysis is made with other such interpretable models like Linear SVM and Logistic Regression with Polynomial Features. These models are also trained over same datasets to bring forth trade-offs between precision and understanding.

## *3.2 Data Generation and Preprocessing*

This research makes use of 2D synthetic classification datasets specifically created to thoroughly assess the model's proficiency in managing various decision boundary types, each symbolising a different classification problem. The datasets have been generated in such a way that there is a balanced representation between two classes, class Red and class Blue.

### 3.2.1 Data Generation for Linear Decision Boundary (Parallel to one of the axis)

To evaluate the model's capability in solving a linear decision boundary problem where the input features $(X_1, X_2)$ which belong to either class 0 or 1 are spread across in two-dimensional feature space in such a way that they could be separated by a straight decision boundary which is parallel to one of the axis, we created synthetic classification data.

**Pseudocode for data generation for linear decision boundary:**

```
1. INITIALISE Parameters:
      - Define the number of data points $(N)$ to be generated.
      - Set the range for the random sampling of features $X_1$ and $X_2$ (e.g.,
$[1-,1]$).
      - Define a and b coefficients for deriving the new feature.
            - a = 1
            - b = 0 (since the decision boundary is parallel to Y-axis, the
      equation will depend solely on a)

2. GENERATE Random Data:
      - Create an array $X$ of size $(N,2)$ where each element is uniformly
sampled from the range $[1-,1]$.
      - Let $X_1$ be the first column of $X$ and $X_2$ be the second column of $X$.

3. ASSIGN Class Labels:
      - For each data point $(X_{1,i}, X_{2,i})$ in $X$:
            - If $(a * X_{1,i} + b * X_{2,i}) > 0$, where $a = 1$ and $b = 0$:
                  - If $(X_{1,i}) > 0$, assign label $y_i = 1$ (Class 1).
                  - Otherwise, assign label $y_i = 0$ (Class 0).

4. VISUALISE Data:
      - Plot the generated data points on a 2D scatter plot.
      - Colour-code the points based on their assigned class labels.
      - Add labels to the x-axis $(X_1)$ and y-axis $(X_2)$ and display the plot.

End of Pseudocode.
```

**Initialisation of Parameters:** The data generation begins by setting the parameters for the dataset. We define $N = 100$, representing the total number of data points to be generated. The features $X_1$ and $X_2$ are randomly sampled from the range $[-1,1]$, ensuring that the data points are evenly distributed across the feature space, covering both positive and negative values.

**Random Data Generation:** Using the "numpy" library, an array of size $(100,2)$ is created, where each element is uniformly sampled from the range $[-1,1]$. This process can be mathematically represented as

$$X_{i,j} \sim U(-1,1), \quad for\ i = 1,2,\dots,100\ and\ j = 1,2$$

where,

$X_{i,j}$ represents the $j-th$ feature of the $i-th$ data point

$U(-1,1)$ denotes a uniform distribution over the interval $[-1,1]$

**Assigning class labels:** After generating the feature values, class labels are assigned based on a simple condition involving the sum of $X_1$ and $X_2$ which should ideally be $a * X_1 + b * X_2$ but since $a = 1$ and $b = 0$, we get to decide the class label solely on the basis of $X_1$.

For each data point $(X_{1,i}, X_{2,i})$:

If $X_{1,i} > 0$ then $y_i = 1$ (Class = 1)

Otherwise, $y_i = 0$ (Class = 0)

This rule ensures that the data points are categorised into two distinct classes so that they can be separated by a linear line parallel to one of the axis.

**Visualisation of data:** As shown in the below figure, the generated data is visualised using a 2D scatter plot to illustrate the distribution of data points across the two classes. The x-axis represents $X_1$ and the y-axis represents $X_2$. The points are colour-coded based on their assigned class labels (e.g., blue for Class 0 and red for Class 1).

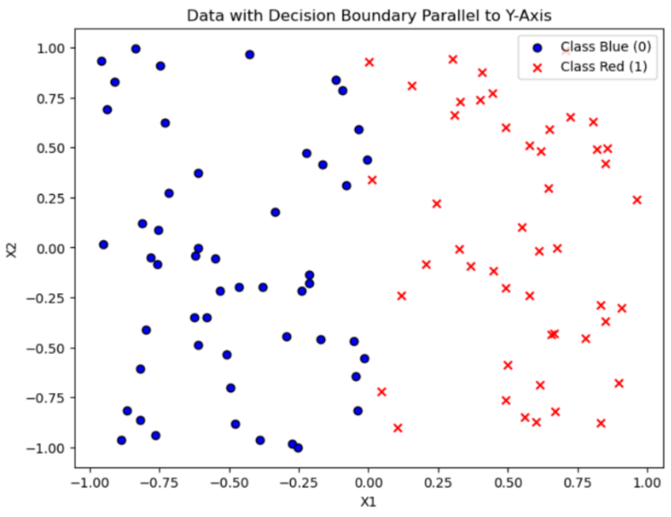

**Figure 3.1:** Scatter plot for data based on decision boundary parallel to Y-Axis

### *3.2.2 Data Generation for Oblique Decision Boundary*

To evaluate the model's capability in solving an oblique decision boundary problem where the input features $(X_1, X_2)$ which belong to either class 0 or 1 are spread across in two-dimensional feature space in such a way that they could be separated by an oblique (linear with slope) decision boundary, we created synthetic classification data.

**Pseudocode for data generation for oblique decision boundary:**

```
1. INITIALISE Parameters:
      - Define the number of data points $(N)$ to be generated.
      - Set the range for the random sampling of features $X_1$ and $X_2$ (e.g.,
$[1-,1]$).
      - Define a and b coefficients for deriving the new feature.
            - a = 1
            - b = 1

2. GENERATE Random Data:
      - Create an array $X$ of size $(N,2)$ where each element is uniformly
sampled from the range $[1-,1]$.
      - Let $X_1$ be the first column of $X$ and $X_2$ be the second column of $X$.

3. ASSIGN Class Labels:
      - For each data point $(X_{1,i}, X_{2,i})$ in $X$:
            - If $(a * X_{1,i} + b * X_{2,i}) > 0$, where $a = 1$ and $b = 1$:
```

```
                              - If $(X_{1,i} + X_{2,i}) > 0$, assign label $y_i = 1$ (Class 1).
                              - Otherwise, assign label $y_i = 0$ (Class 0).

4. VISUALISE Data:
        - Plot the generated data points on a 2D scatter plot.
        - Colour-code the points based on their assigned class labels.
        - Add labels to the x-axis $(X_1)$ and y-axis $(X_2)$ and display the plot.

End of Pseudocode.
```

**Initialisation of Parameters:** The data generation begins by setting the parameters for the dataset. We define $N = 100$, representing the total number of data points to be generated. The features $X_1$ and $X_2$ are randomly sampled from the range $[-1,1]$, ensuring that the data points are evenly distributed across the feature space, covering both positive and negative values.

**Random Data Generation:** Using the "numpy" library, an array of size $(100,2)$ is created, where each element is uniformly sampled from the range $[-1,1]$. This process can be mathematically represented as

$$X_{i,j} \sim U(-1,1), \quad for\ i = 1,2,\ldots,100\ and\ j = 1,2$$

where,

$X_{i,j}$ represents the $j - th$ feature of the $i - th$ data point

$U(-1,1)$ denotes a uniform distribution over the interval $[-1,1]$

**Assigning class labels:** After generating the feature values, class labels are assigned based on a simple condition involving the sum of $X_1$ and $X_2$ which should ideally be $a * X_1 + b * X_2$ but since $a = 1$ and $b = 1$, we get to decide the class label on the basis of $X_1 + X_2$.

For each data point $(X_{1,i}, X_{2,i})$:

If $X_{1,i} + X_{2,i} > 0$ then $y_i = 1$ (Class = 1)

Otherwise, $y_i = 0$ (Class = 0)

This rule ensures that the data points are categorised into two distinct classes so that they can be separated by an oblique decision boundary.

**Visualisation of data:** As shown in the below figure, the generated data is visualised using a 2D scatter plot to illustrate the distribution of data points across the two classes. The x-axis represents $X_1$ and the y-axis represents $X_2$. The points are colour-coded based on their assigned class labels (e.g., blue for Class 0 and red for Class 1).

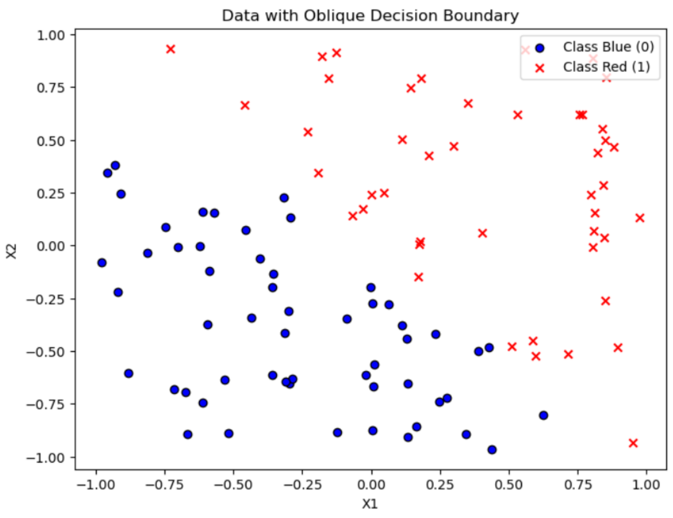

**Figure 3.2:** Scatter plot for data based on oblique decision boundary

### *3.2.3 Data Generation for Circular Decision Boundary*

Similarly, to evaluate the model's capability in solving a circular classification problem, we created synthetic data with a simple circular structure. The idea was to form two classes of data which would be separated in a two-dimensional space such that there would be a circular decision boundary.

**Pseudocode for data generation for circular decision boundary:**

```
1. INITIALISE Parameters:
        - Define the number of data points $(N)$ to be generated.
        - Set the range for the random sampling of features $X_1$ and $X_2$ (e.g.,
$[1-,1]$).

2. GENERATE Random Data:
        - Create an array $X$ of size $(N,2)$ where each element is uniformly
sampled from the range $[1-,1]$.
        - Let $X_1$ be the first column of $X$ and $X_2$ be the second column of $X$.

3. ASSIGN Class Labels:
        - For each data point $\left(X_{(1,i)},X_{(2,i)}\right)$ in $X$:
              - If $\left({X^2}_{(1,i)} + {X^2}_{(2,i)}\right) > 0.5$, assign label $y_i = 1$ (Class 1).
              - Otherwise, assign label $y_i = 0$ (Class 0).

4. VISUALISE Data:
        - Plot the generated data points on a 2D scatter plot.
        - Colour-code the points based on their assigned class labels.
        - Add labels to the x-axis $(X_1)$ and y-axis $(X_2)$ and display the plot.

End of Pseudocode.
```

**Initialisation of Parameters:** Similar to data generation for linear decision boundary, the data generation for circular decision boundary begins by setting the parameters for the dataset. We define $N = 100$, representing the total number of data points to be generated. The features $X_1$ and $X_2$ are randomly sampled from the range $[-1,1]$, ensuring that the data points are evenly distributed across the feature space, covering both positive and negative values.

**Random Data Generation:** Using the "numpy" library, an array of size $(100,2)$ is created, where each element is uniformly sampled from the range $[-1,1]$. This process can be mathematically represented as

$$X_{i,j} \sim U(-1,1), \quad for\ i = 1,2,\dots,100\ and\ j = 1,2$$

where,

$X_{i,j}$ represents the $j-th$ feature of the $i$-th data point

$U(-1,1)$ denotes a uniform distribution over the interval $[-1,1]$

**Assigning class labels:** Class labels are assigned based on a circular decision boundary. For each data point $\left(X_{(1,i)}, X_{(2,i)}\right)$, the following condition is checked:

If ${X^2}_{(1,i)} + {X^2}_{(2,i)} > 0.5$ then $y_i = 1$ (Class = 1)

Otherwise, $y_i = 0$ (Class = 0)

This rule classifies two different classes divided by a circular decision boundary. Any point that is not within a radius of $\sqrt{0}.5$ is classified as Class 1, while points that are on or within the circle are classified as Class 0.

**Visualisation of data:** Like linear decision boundary the below figure shows a visualisation of generated data using a 2D scatter plot to illustrate the distribution of data points across the two classes. The x-axis represents $X_1$ and the y-axis represents $X_2$. The points are colour-coded based on their assigned class labels (e.g., blue for Class 0 and red for Class 1).

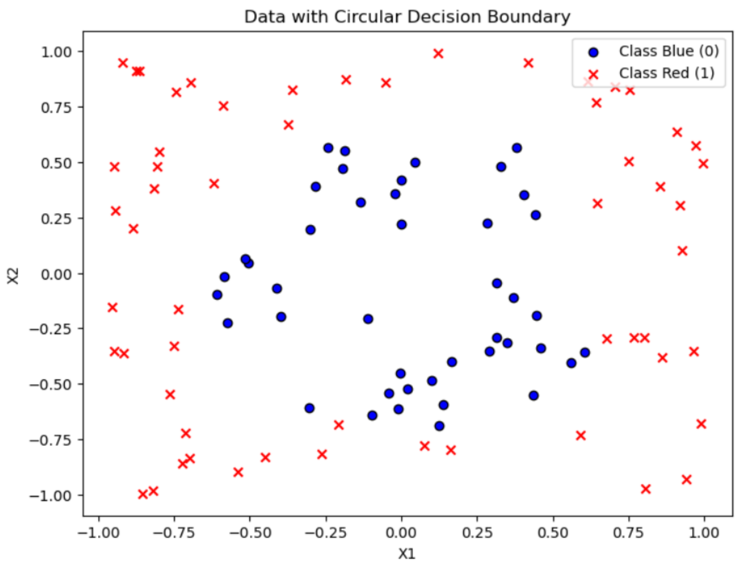

**Figure 3.3:** Scatter plot for data based on circular decision boundary

### 3.2.4 *Data Generation for Elliptical Decision Boundary*

Like the previous section, we developed synthetic data with an elliptical distribution to examine the model's performance on classification problems with elliptical boundaries. This setup was designed to create two classes, clearly divided by a elliptical decision boundary within a two-dimensional space.

**Pseudocode for data generation for elliptical decision boundary:**

```
1. INITIALISE Parameters:
      - Define the number of data points ($N$) to be generated.
      - Set the range for the random sampling of features $X_1$ and $X_2$ (e.g.,
$[1-,1]$).
      - Define the ellipse parameters.
            - Let $a$ be the semi-major axis of the ellipse (e.g., $a = 0.7$)
            - Let $b$ be the semi-minor axis of the ellipse (e.g., $b = 0.5$)


2. GENERATE Random Data:
      - Create an array $X$ of size $(N,2)$ where each element is uniformly
sampled from the range $[1-,1]$.
      - Let $X_1$ be the first column of $X$ and $X_2$ be the second column of $X$.

3. ASSIGN Class Labels:
      - For each data point $\left(X_{(1,i)}, X_{(2,i)}\right)$ in $X$:
            - If $\left(X^2{}_{(1,i)}/a^2 + X^2{}_{(2,i)}/b^2\right) > 1$, assign label $y_i = 1$ (Class 1).
            - Otherwise, assign label $y_i = 0$ (Class 0).

4. VISUALISE Data:
      - Plot the generated data points on a 2D scatter plot.
      - Colour-code the points based on their assigned class labels.
      - Add labels to the x-axis $(X_1)$ and y-axis $(X_2)$ and display the plot.

End of Pseudocode.
```

**Initialisation of Parameters:** Similar to data generation for circular decision boundary, the data generation for elliptical decision boundary begins by setting the parameters for the dataset. We define $N = 100$, representing the total number of data points to be generated. The features $X_1$ and $X_2$ are randomly sampled from the range $[-1,1]$, ensuring that the data points are evenly distributed across the feature space, covering both positive and negative values. Along with this, in case of an elliptical decision boundary we also need to set up semi-major $(a)$ axis and semi-minor $(b)$ axis parameters which for an ideal ellipse are $a = 0.7$ and $b = 0.5$.

**Random Data Generation:** Using the "numpy" library, an array of size $(100,2)$ is created, where each element is uniformly sampled from the range $[-1,1]$. This process can be mathematically represented as

$$X_{i,j} \sim U(-1,1), \quad for\ i = 1,2,\dots,100\ and\ j = 1,2$$

where,

$X_{i,j}$ represents the $j-th$ feature of the $i$-th data point

$U(-1,1)$ denotes a uniform distribution over the interval $[-1,1]$

**Assigning class labels:** Class labels are assigned based on a elliptical decision boundary. For each data point $\left(X_{(1,i)}, X_{(2,i)}\right)$, the following condition is checked:

If $X^2{}_{(1,i)}/a^2 + X^2{}_{(2,i)}/b^2 > 1$ then $y_i = 1$ (Class = 1)

Otherwise, $y_i = 0$ (Class = 0)

**Visualisation of data:** Like circular decision boundary the below figure shows a visualisation of generated data using a 2D scatter plot to illustrate the distribution of data points across the two classes. The x-axis represents $X_1$ and the y-axis represents $X_2$. The points are colour-coded based on their assigned class labels (e.g., blue for Class 0 and red for Class 1).

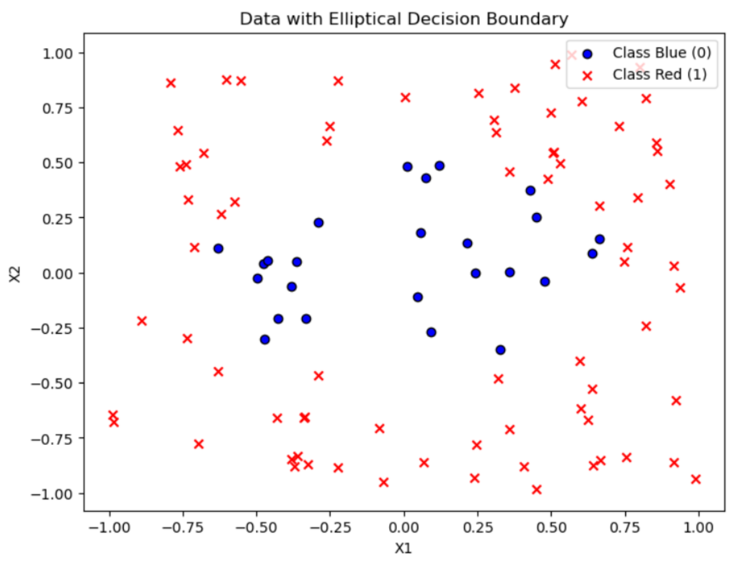

**Figure 3.4:** Scatter plot for data based on elliptical decision boundary

### *3.2.5 Data Generation for Square Decision Boundary*

Following the same approach as before, we created synthetic data arranged in a square pattern to evaluate the model's effectiveness in solving classification problems with square boundaries. This setup aimed to differentiate two data classes within a 2D space using a square decision boundary.

**Pseudocode for data generation for square decision boundary:**

```
1. INITIALISE Parameters:
      - Define the number of data points $(N)$ to be generated.
      - Set the range for the random sampling of features $X_1$ and $X_2$ (e.g.,
$[1-,1]$).
      - Define the threshold value for the square decision boundary.

2. GENERATE Random Data:
      - Create an array $X$ of size $(N,2)$ where each element is uniformly
sampled from the range $[1-,1]$.
      - Let $X_1$ be the first column of $X$ and $X_2$ be the second column of $X$.

3. ASSIGN Class Labels:
      - For each data point $\left(X_{(1,i)},X_{(2,i)}\right)$ in $X$:
            - If $max\left(|X_{(1,i)}|,|X_{(2,i)}|\right) > threshold$, assign label $y_i = 1$ (Class 1).
            - Otherwise, assign label $y_i = 0$ (Class 0).

4. VISUALISE Data:
      - Plot the generated data points on a 2D scatter plot.
      - Colour-code the points based on their assigned class labels.
      - Add labels to the x-axis $(X_1)$ and y-axis $(X_2)$ and display the plot.


End of Pseudocode.
```

**Initialisation of Parameters:** Similar to data generation for elliptical decision boundary, the data generation for square decision boundary begins by setting the parameters for the dataset. We define $N = 100$, representing the total number of data points to be generated. The features $X_1$ and $X_2$ are randomly sampled from the range $[-1,1]$, ensuring that the data points are evenly distributed across the feature space, covering both positive and negative values.

**Random Data Generation:** Using the "numpy" library, an array of size $(100,2)$ is created, where each element is uniformly sampled from the range $[-1,1]$. This process can be mathematically represented as

$$X_{i,j} \sim U(-1,1), \quad for\ i = 1,2, \ldots, 100\ and\ j = 1,2$$

where,

$X_{i,j}$ represents the $j-th$ feature of the $i$-th data point

$U(-1,1)$ denotes a uniform distribution over the interval $[-1,1]$

**Assigning class labels:** Class labels are assigned based on a square decision boundary. For each data point $\left(X_{(1,i)}, X_{(2,i)}\right)$, the following condition is checked:

If $max\left(|X_{(1,i)}|, |X_{(2,i)}|\right) > threshold$ then $y_i = 1$ (Class = 1)

Otherwise, $y_i = 0$ (Class = 0)

Here, the threshold is set to a specific value (e.g., 0.5), which defines the square decision boundary. This rule creates a square-shaped classification boundary centered at the origin. Points lying outside this square boundary are classified as Class 1, while points within or on the boundary are classified as Class 0.

**Visualisation of data:** Like elliptical decision boundary the below figure shows a visualisation of generated data using a 2D scatter plot to illustrate the distribution of data points across the two classes. The x-axis represents $X_1$ and the y-axis represents $X_2$. The points are colour-coded based on their assigned class labels (e.g., blue for Class 0 and red for Class 1).

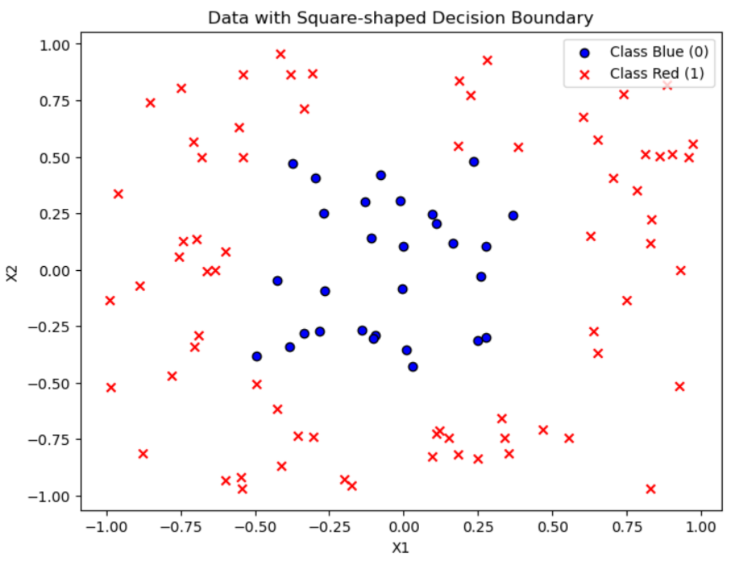

**Figure 3.5:** Scatter plot for data based on square decision boundary

## 3.3 Feature Engineering: Introduction of Derived Feature

Feature engineering is a crucial step in the Machine Learning process. It is the process of developing new features or modifying existing features with the aim of improving machine learning models. The main objective of feature engineering is to pull out the usable amount of information from raw data such that the model can capture the core methods effectively. In this project, a derived feature more like a vector, is created concerning each type of decision boundary (linear, oblique, circular, elliptical and square) for the purpose of enhancing how a decision tree interprets those decision boundaries of a neural network. This feature is defined using different metrics, which in this case is defined by the classification dataset but is intended for use by the decision tree as a way of helping to replicate the decision-making of the neural network. The neural network itself does not utilise or have any knowledge of this feature.

### 3.3.1 Generation of Z for Linear Decision Boundary (Parallel to one of the Axis)

To interpret the linear decision boundary which is parallel to one of the axis (in our case Y-axis) based on the underlying data, we introduce a feature $Z_{lin}$. This derived feature is a linear combination of input features $X_1$ and $X_2$. The aim is to create a single feature or a vector that will capture the pattern in the underlying data, simplifying the task of decision tree to mimic neural network's decisions.

The derived feature $Z_{lin}$ is calculated as follows:

$$Z_{lin} = a * X_1 + b * X_2$$

where,

$a$ and $b$ are the coefficients that define the linear combination.
$X_1$ and $X_2$ are the original input features

In this case, we set $a = 1$ and $b = 0$ since the decision boundary is parallel to Y-axis, the feature will solely rely on $X_1$. So, we get, $Z_{lin} = a * X_1 = X_1$.

This derived feature $Z_{lin}$ is used solely by the decision tree to approximate the decision-making process of the neural network. The neural network does not have any idea about this feature, instead it continues to train itself on the original features $X_1$ and $X_2$.

### 3.3.2 Generation of Z for Oblique Decision Boundary

To interpret the linear decision boundary generated by the neural network for underlying data, we introduce a derived feature $Z_{linear}$. This derived feature is a linear combination of input features $X_1$ and $X_2$. The aim is to create a single feature that will capture the pattern in the underlying data, simplifying the task of decision tree to approximate neural network's decisions.

The derived feature $Z_{linear}$ is calculated as follows:

$$Z_{linear} = a * X_1 + b * X_2$$

where,

$a$ and $b$ are the coefficients that define the linear combination.
$X_1$ and $X_2$ are the original input features

In this case, we set $a = 1$ and $b = 1$ because the decision boundary has slope attributed with it, so we get $Z_{linear} = X_1 + X_2$.

This derived feature $Z_{linear}$ is used solely by the decision tree to approximate the decision-making process of the neural network. The neural network does not have any idea about this feature, instead it continues to train itself on the original features $X_1$ and $X_2$.

### 3.3.3 Generation of Z for Circular Decision Boundary

Similarly, to interpret the circular decision boundary generated by the neural network for the underlying data, we introduce a derived feature $Z_{circular}$. This feature is designed to capture the circular pattern in the data by using the squared values of the input features $X_1$ and $X_2$. The objective is to simplify the decision tree's task to approximate neural network's decisions.

The derived feature $Z_{circular}$ is calculated as follows:

$$Z_{circular} = X^2{}_1 + X^2{}_2$$

where,

$X_1$ and $X_2$ are the original input features

This feature effectively represents the squared Euclidean distance from the origin, providing a measure of how far each point lies from the center of the circular decision boundary.

### 3.3.4 Generation of Z for Elliptical Decision Boundary

Similarly, to interpret the elliptical decision boundary generated by the neural network for the underlying data, we introduce a derived feature $Z_{ellipse}$. This feature is constructed by transforming the input features $X_1$ and $X_2$ according to an elliptical equation. The purpose is to distill the complex, non-linear elliptical relationship into a single feature that the decision tree can use to better mimic the neural network's decision-making process.

The derived feature $Z_{ellipse}$ is calculated as follows:

$$Z_{ellipse} = X^2{}_1/a^2 + X^2{}_2/b^2$$

where,

$a$ and $b$ represent the semi-major and semi-minor axes of the ellipse, respectively.
$X_1$ and $X_2$ are the original input features

This feature captures the extent to which each data point deviates from the center of the ellipse, normalised by the axes' lengths.

### 3.3.5 Generation of Z for Square Decision Boundary

In a similar way, to enhance the interpretability of the neural network's understanding of a square-shaped decision boundary, we derive a feature $Z_{square}$. This feature is created by applying a transformation to the input features $X_1$ and $X_2$ that reflects the square’s sharp edges.

The derived feature $Z_{square}$ is computed as:

$$Z_{square} = max(|X_1|, |X_2|)$$

Where,

$X_1$ and $X_2$ are the original input features

This feature represents the maximum absolute deviation of each data point from the origin along any axis, capturing the square's boundary.

Now that we have generated the required data, in the upcoming section we will deep dive into model training and evaluation.

## 3.4 Model Training and Evaluation

In machine learning, training the model is a fundamental process where data is used to teach the algorithm to identify patterns and make decisions. This stage involves the adjustment of the algorithm's parameters to decrease the errors in its predictions on data it has not seen before, thus enhancing its ability to apply its learnings to broader datasets. The training of the model is vital for capturing the essential data structures and relationships, which are necessary for the algorithm to perform accurately and reliably.

### 3.4.1 Neural Network Model Training

Neural networks, powerful machine learning tools modelled after the human brain, are adept at capturing complex patterns and relationships in data. In this research, the Multi-Layer Perceptron (MLP) classifier from the scikit-learn library is used to train neural networks on various datasets, each with unique decision boundaries like linear, circular, elliptical, and square. We design the neural network with a single hidden layer containing 20 neurons, ensuring it is sufficiently complex to grasp the specifics of each dataset while avoiding overfitting.

**Pseudocode for Neural Network Model Training:**

```
1. INITIALISE Neural Network Model:
      - Use the Multi-Layer Perceptron (MLP) classifier with:
            - $hidden\_layer\_sizes = (20,)$ (one hidden layer with 20 neurons)
            - $max\_iter = 1000$ (maximum number of iterations for training)
            - $random\_state = 42$ (fixed seed for reproducibility)

2. TRAIN Neural Network for each dataset:
      - Train the neural network on each dataset (linear, circular, elliptical, square):
            - $nn\_model\_linear.fit(X\_linear, y\_linear)$
            - $nn\_model\_circular.fit(X\_circular, y\_circular)$
            - $nn\_model\_ellipse.fit(X\_ellipse, y\_ellipse)$
            - $nn\_model\_square.fit(X\_square, y\_square)$

3. GENERATE Neural Network Predictions for each dataset:
      - Store the neural network predictions for each dataset (linear, circular, elliptical, square):
            - $y\_nn\_pred\_linear = nn\_model\_linear.predict(X\_linear)$
            - $y\_nn\_pred\_circular = nn\_model\_circular.predict(X\_circular)$
            - $y\_nn\_pred\_ellipse = nn\_model\_ellipse.predict(X\_ellipse)$
            - $y\_nn\_pred\_square = nn\_model\_square.predict(X\_square)$

End of Pseudocode.
```

The inputs to the neural network will be the features from the datasets. For example, in the linear, circular, elliptical, and square datasets, each data point has two primary features, $X_1$ and $X_2$ representing a point in 2D space. Therefore, the neurons in the input layer will take these features as inputs. Since each dataset contains two features, the input layer will consist of 2 neurons corresponding to these features.

The neural network will have 1 hidden layer consisting of 20 hidden neurons. Each neuron in this hidden layer will receive inputs from all neurons in the input layer. Each hidden neuron will also have a weight vector associated with the inputs (one weight per input feature) and a bias term. The weighted sum of inputs and the bias will be passed through an activation function sigmoid to produce an output.

The output $h_i$ of a hidden neuron $i$ can be expressed as:

$$h_i = f(w_{i1} * X_1 + w_{i2} * X_2 + b_i)$$

where,

$w_{i1}$ and $w_{i2}$ are the weights corresponding to inputs $X_1$ and $X_2$
$b_i$ is the bias for neuron $i$
$f$ is the activation function which in this case will be a sigmoid function

Since this is a binary classification problem (Class 0 or Class 1), the output layer will have 1 neuron that provides a single output value representing the probability of the input belonging to Class 1. The output neuron will be connected to each neuron in the hidden layer, with its own set of weights and a bias term.

The output neuron will typically use a sigmoid activation function to map the output to a probability value between 0 and 1:

$$y = \sigma(\sum_{i=1}^{20} w_{o,i} * h_i + b_o)$$

where,

$w_{o,i}$ are the weights connecting the hidden neurons to the output neuron,
$h_i$ are the outputs from the hidden neurons,
$b_o$ is the bias term for the output neuron,
σ is the sigmoid activation function which is defined as:

$$\sigma(x) = \frac{1}{1 + \varepsilon^{-x}}$$

Below is the illustration of the structure of the neural network specific to our research:

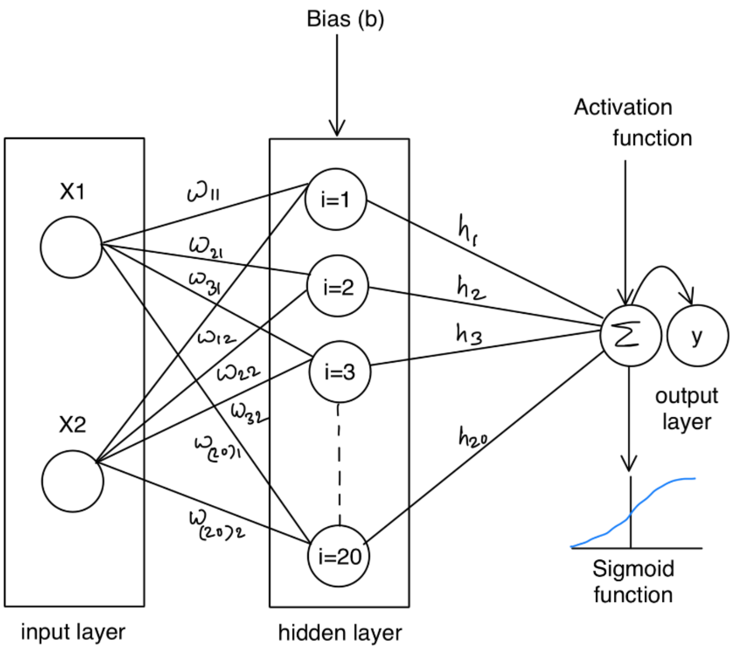

**Figure 3.6:** Structure of the Neural Network

Using the $\boldsymbol{fit}()$ function, the network's weights and biases are adjusted via backpropagation to minimise the loss function, ensuring precise classification of the data points. Importantly, the neural network delineates the decision boundaries for each dataset based solely on the original features, without any influence from the derived feature, maintaining the purity of the network's decision-making process.

### 3.4.2 Decision Tree Model Training

Decision trees are models that make a series of hierarchical decisions to classify data points, using input features. This study involves training decision trees to mimic the decision boundaries established by neural networks. The decision trees use the outputs of the neural networks as their target labels. Each dataset is used to train two models of decision trees: one model incorporates the original features along with the derived feature, and the second model is trained without the derived feature.

**Pseudocode for Decision Tree Model Training:**

```
1. COMBINE Features with Derived Feature Z:
      - For each dataset: Concatenate original features with the derived feature $Z$:
            - $X\_combined\_linear = np.column\_stack((X\_linear, Z\_linear))$
            - $X\_combined\_circular = np.column\_stack((X\_circular, Z\_circular))$
            - $X\_combined\_ellipse = np.column\_stack((X\_ellipse, Z\_ellipse))$
            - $X\_combined\_square = np.column\_stack((X\_square, Z\_square))$

2. TRAIN Decision Trees:
      - Train decision trees on the combined datasets and Neural Networks' Predictions using feature Z i.e. (X1, X2, Z):
            - $tree\_model\_linear.fit(X\_combined\_linear, y\_nn\_pred\_linear)$
            - $tree\_model\_circular.fit(X\_combined\_circular, y\_nn\_pred\_circular)$
            - $tree\_model\_ellipse.fit(X\_combined\_ellipse, y\_nn\_pred\_ellipse)$
            - $tree\_model\_square.fit(X\_combined\_square, y\_nn\_pred\_square)$

      - Train decision trees on the original datasets and Neural Networks' Predictions but without using the derived feature $Z$ i.e. (X1, X2):
            - $tree\_model\_linear\_without\_z.fit(X\_linear, y\_nn\_pred\_linear)$
            - $tree\_model\_circular\_without\_z.fit(X\_circular, y\_nn\_pred\_circular)$
            - $tree\_model\_ellipse\_without\_z.fit(X\_ellipse, y\_nn\_pred\_ellipse)$
            - $tree\_model\_square\_without\_z.fit(X\_square, y\_nn\_pred\_square)$

End of Pseudocode.
```

Training decision trees in this study consists of two primary phases: first, retrieving predictions from the neural networks, and second, using these predictions along with the input data to train the decision trees to mirror the neural networks' decision-making framework. The decision trees are trained using both sets of features; with and without the derived feature, in order to determine how the addition of the derived feature helps decision trees to enhance their capability understand the data and to mimic the classification done by the Neural Network without sacrificing model interpretability and overall performance. Results from this approach are instrumental in assessing whether an extra feature which provides the essence of the underlying data plays a role in simplifying the decision boundary as learned by the neural network.

### 3.4.3 Comparison with other Interpretable Machine Learning Models

To deepen our understanding of the interpretability and performance of the decision boundaries created by the neural networks and decision trees, we conduct comparisons with well-established machine learning models like Linear Support Vector Machine (SVM) for classification of data having linear and oblique decision boundaries and Logistic Regression with polynomial features for classification of data having circular decision boundary. These comparisons serve as benchmarks, providing insights into the complexity of the decision boundaries.

**Linear Support Vector Machine (SVM):** Linear SVM is a supervised learning model primarily used for classification tasks. It operates by identifying the most suitable hyperplane that distinctly separates the data points of different classes within a high-dimensional space. For linear SVM, this hyperplane is a straight line that is strategically positioned to maximise the distance between the two classes. The points nearest to the hyperplane, known as support vectors, are pivotal in determining both the position and the orientation of the hyperplane. The figure below illustrates how a linear SVM classifies two distinct classes of data.

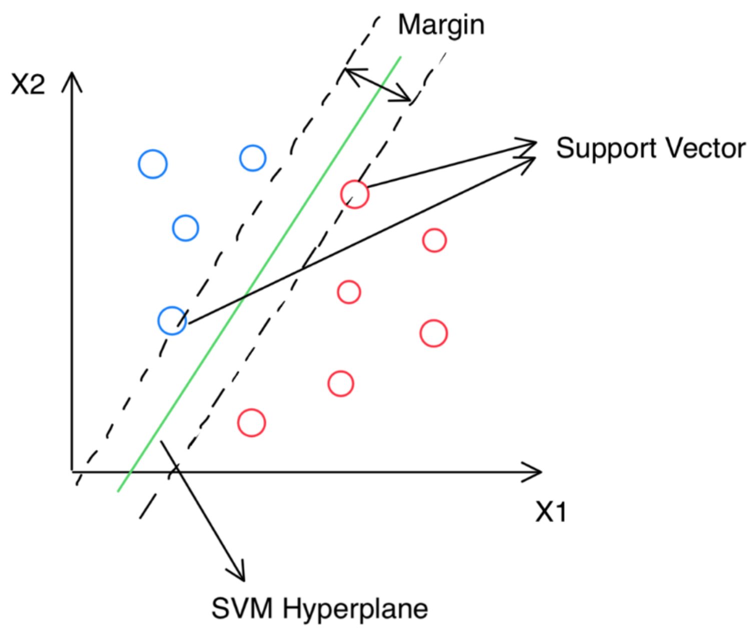

**Figure 3.7:** Structure of Linear SVM

Linear SVM is favoured for model interpretability due to its straightforward geometric boundary that distinguishes classes, which is expressible through a mathematical formula. Nevertheless, formulating a single equation to describe this hyperplane can be complex, particularly in scenarios involving high-dimensional spaces or numerous support vectors, since the hyperplane relies on a blend of support vectors and their respective weights rather than on a simple equation alone.

**Pseudocode for Training Linear SVM on data with Linear as well as Oblique Decision Boundaries:**

```
1. INITIALISE Linear SVM Model:
      - Use a Linear Support Vector Classifier (SVC) with:
            - $kernel = 'linear'$ (to define a linear hyperplane)

2. TRAIN Linear SVM:
      - Train the SVM on the combined dataset with derived feature $Z$
            - $svm\_model\_lin.fit(X\_combined\_lin, y\_nn\_pred\_lin)$ (for linear decision
            boundary)
            - $svm\_model\_linear.fit(X\_combined\_linear, y\_nn\_pred\_linear)$ (for oblique
            decision boundary)
```

```
3. PREDICT Using Linear SVM:
      - Predict class labels using the trained SVM model:
            - $y\_pred\_svm\_lin = svm\_model\_lin.predict(X\_combined\_lin)$ (for linear
            decision boundary)
            - $y\_pred\_svm = svm\_model\_linear.predict(X\_combined\_linear)$ (for oblique
            decision boundary)

End of Pseudocode.
```

The linear SVM is trained using the combined dataset, which incorporates the derived feature $Z_{lin}$ in case of linear data with the decision boundary parallel to the Y-Axis and $Z_{linear}$ in case of oblique decision boundary, to evaluate its effectiveness in approximating the decision boundaries identified by the neural network in both the cases. Employing a linear kernel, the SVM strives to identify an optimal hyperplane that distinctly separates the data points. Although the SVM's decision boundary is straightforward to visualise and interpret, deriving a singular equation for the hyperplane is complex due to its reliance on multiple support vectors. This dependency complicates providing a succinct mathematical explanation of the SVM's decision-making process.

**Logistic Regression with Polynomial Features:** Logistic Regression is a linear model that is primarily employed for binary classification purposes. It estimates the likelihood that a specific input belongs to a certain class by applying a linear equation to the dataset. For complex, non-linear decision boundaries, the inclusion of polynomial features in Logistic Regression proves beneficial. Polynomial features expand the model's capacity to capture complex interactions between variables by generating higher-order terms, thereby allowing it to accurately represent curved decision boundaries.

In this research, Logistic Regression equipped with polynomial features is employed to effectively model the non-linear, circular decision boundary associated with the underlying data. The incorporation of a second-degree polynomial transformation enables the model to learn and understand quadratic relationships among the features, which is crucial for identifying the boundary.

**Pseudocode for Training Logistic Regression with Polynomial Features:**

```
1. INITIALISE Logistic Regression Model with Polynomial Features:
      - Create a pipeline that includes:
            - $PolynomialFeatures(degree = 2)$ (to generate polynomial features up
to degree 2)
            - $LogisticRegression()$ (to perform logistic regression on the
transformed features)

2. TRAIN Logistic Regression:
      - Train the model on the combined dataset
            - $poly\_logistic.fit(X\_combined\_circular, y\_circular)$

3. PREDICT Using Logistic Regression:
      - Predict class labels using the trained model:
            - $y\_pred\_poly\_logistic = poly\_logistic.predict(X\_combined\_circular)$

End of Pseudocode.
```

Through the $PolynomialFeatures$ transformation, the input features are extended into a higher-dimensional space, which facilitates the logistic regression model in learning a quadratic decision boundary that aligns with the circular characteristics of the dataset. Although the addition of polynomial features improves the model's handling of non-linear relationships, it can also escalate the complexity of the model and lessen its interpretability with higher polynomial degrees.

In the next section, we will talk about different evaluation metrics used to test the accuracy of the models.

### 3.5 Evaluation Metrics

In machine learning, training the model is a fundamental process where data is used to teach the algorithm to identify patterns and make decisions. This stage involves the adjustment of the algorithm's parameters to decrease the errors in its predictions on data it has not seen before, thus enhancing its ability to apply its learnings to broader datasets. The training of the model is vital for capturing the essential data structures and relationships, which are necessary for the algorithm to perform accurately and reliably.

In this research, the primary evaluation metrics used are accuracy, precision, recall, F1-score, and ROC-AUC. Accuracy stands as the foundational metric, quantifying the ratio of correctly predicted instances to the total dataset. However, relying solely on accuracy may overlook critical aspects of model performance, especially under conditions of class imbalance or where certain errors are more significant than others. Thus, precision and recall are crucial metrics as well.

Precision reflects the accuracy of positive predictions, and recall indicates the effectiveness of the model in identifying all positives within the dataset. To synthesize these insights, the F1-score is used, representing a harmonic mean that helps balance the precision-recall trade-off, essential in managing unbalanced datasets.

### 3.6 Summary

To summarise, the overall Methodology section talked in detail about how different synthetic classification datasets having different decision boundaries like linear, oblique, circular, elliptical and square-shaped were created. The methodology elaborated on deriving a distinctive feature for each dataset to help reveal essential data patterns and support decision trees in approximating the decision boundaries that neural networks learn. Neural networks were trained to establish these boundaries, followed by the training of decision trees, using both scenarios involving the derived feature and without, to check their precision in reflecting neural network decisions. The methodology also compares these models with traditional counterparts like Linear SVM and Logistic Regression with polynomial features. These models were rigorously evaluated using metrics such as accuracy, precision, recall, F1-score, and ROC-AUC to comprehensively measure their performance and enhance interpretability in line with Explainable Artificial Intelligence (XAI) objectives.

# Chapter 4: Experiments and Results

## *4.1 Experimental Setup*

The configuration of the experiments in this study is crafted to ensure reproducibility, validity, and alignment with the objectives of enhancing the interpretability of machine learning models by integrating decision trees with neural networks. The experimental process starts by training, testing, and critically evaluating different machine learning models on synthetic classification datasets designed with a variety of geometric decision boundaries, such as linear, oblique, circular, elliptical, and square. Detailed in this section are the computational settings, the methodology for preparing the data, the model training configurations, and the testing protocols implemented in this research.

**Computational Environment:** All experiments were conducted in a Python environment using a combination of widely recognised libraries to facilitate data generation, model training, and visualisation. Below is a detailed list of the tools and libraries used:

1. **Programming Language:** Python
2. **Development Environment:** Jupyter Notebook
3. **Hardware Specifications:**
    - Processor: Intel Core i7 CPU
    - RAM: 16 GB
    - Operating System: Windows 10
4. **Primary Libraries and Tools:**
    - **NumPy (version 1.19):** For numerical operations, data manipulation, and generation of synthetic datasets.
    - **Matplotlib (version 3.3):** For plotting and visualisations, including decision boundaries and data distribution.
    - **scikit-learn (version 0.24):** For implementing machine learning models such as:
        - $MLPClassifier$ for neural networks.
        - $DecisionTreeClassifier$ for decision trees.
        - $LogisticRegression$ for logistic regression with polynomial features.
        - $SVC$ for Support Vector Machines (SVMs).
        - $Pipeline$ for creating a pipeline for logistic regression with polynomial features.
        - $PolynomialFeatures$ for transforming data to include polynomial features.
        - $Metrics$ such as $accuracy\_score$, $precision\_score$, $recall\_score$, $f1\_score$ and $roc\_auc\_score$.
    - **pydotplus:** For creating visualisations of decision trees in conjunction with $graphviz$.
    - **graphviz:** For exporting and rendering decision trees into graphical formats.
    - **IPython.display(Image):** For displaying images of decision trees in Jupyter Notebook.

This setup ensured sufficient computational resources and provided all necessary tools to implement, train, evaluate, and visualise the machine learning models effectively without requiring GPU acceleration.

**Data Preparation:** To evaluate the interpretability of neural networks combined with decision trees, synthetic classification datasets each containing 100 $(x, y)$ pairs where each pair is labelled as either Class 0 or Class 1 based on different geometric decision boundaries that separates one class of points with another. We have considered cases where the decision boundary could represent shapes like linear, oblique, circular, elliptical, and square for this research. Each dataset consists of two primary features $X_1$ and $X_2$, uniformly sampled from the range $[-1, 1]$ to ensure balanced representation across different classes.

- **Data Generation:** For each type of decision boundary, a dataset of 100 data points was created using $numpy$'s $uniform$ function. The class labels were assigned based on specific geometric rules:
  - **Linear Data:** Points were labelled according to whether $X_1$ was greater than zero or not since the decision boundary is parallel to the Y-Axis it's equation will not rely on $X_2$.
  - **Oblique Data:** Points were labelled according to whether the sum of $X_1$ and $X_2$ was greater than zero.
  - **Circular Data:** Labels were assigned based on whether the sum of squares $X_1^2 + X_2^2$ exceeded a threshold (e.g., 0.5).
  - **Elliptical Data:** Points were classified using an elliptical decision boundary $\left(\frac{X_1^2}{a^2} + \frac{X_2^2}{b^2} > 1\right)$ with $a = 0.7$ and $b = 0.5$.
  - **Square Data:** Labels were determined by checking whether the maximum absolute value of the coordinates $(\max(|X_1|, |X_2|))$ was greater than a set threshold (e.g., 0.5).

No explicit splitting of data into separate training and testing sets was performed. Instead, the entire dataset was used for training the neural networks and using the neural networks' predictions as targets for decision trees we evaluate their performance and interpretability. This approach allowed for a consistent assessment of model performance across all data points and ensured that all data was utilised to evaluate the decision trees' capability to approximate the neural networks' decision boundaries.

**Model Training:** The training process involves using neural networks to establish decision boundaries for different types of classification data and subsequently training the decision trees using neural networks' predictions along with the input features $(X_1, X_2)$ to enhance the capability to approximate these boundaries. The goal is to create interpretable decision trees that closely mimic the behaviour of neural networks while leveraging a derived feature that captures the underlying geometric patterns of the data. This approach can be generalised to any two-class classification problem by appropriately defining the feature based on the nature of the decision boundary as of now.

1. **Neural Network Training:** For each dataset, a neural network is trained to learn the decision boundary:
   - **Neural Network Configuration:**
     - **Model:** Multi-Layer Perceptron (MLP) classifier from $scikit - learn$.
     - **Architecture:** Single hidden layer with 20 neurons and ReLU activation function.
     - **Training Parameters:**
       - **Optimizer:** Adam
       - **Maximum iterations:** 1000
       - **Random state:** 42 (to ensure reproducibility)
   - **Training Process:**
     - Load the dataset consisting of two features $X_1$ and $X_2$
     - Train the MLP classifier using the entire dataset to learn the decision boundary.
   - **Output:**
     - The neural network is trained to predict the class labels of the input data, establishing a decision boundary that separates the two classes.
2. **Decision Tree Training:** Decision trees are trained to approximate the decision boundaries established by the neural networks, with and without the derived feature Z.
   - **Determine Feature Z:**

- Define the feature Z based on the geometric nature of the decision boundary:
  - For linear decision boundary: $Z = X_1$
  - For oblique decision boundary: $Z = X_1 + X_2$
  - For circular decision boundary: $Z = X_1^2 + X_2^2$
  - For elliptical decision boundary: $Z = \frac{X_1^2}{a^2} + \frac{X_2^2}{b^2}$ where $a$ and $b$ are the semi-major and semi-minor axes.
  - For square-shaped decision boundary: $Z = \max(|X_1|, |X_2|)$

- **Decision Tree Configuration:**
  - **Model:** DecisionTreeClassifier from $scikit - learn.$
  - **Parameters:**
    - Maximum depth: 1 (to ensure interpretability)
    - Criterion: Gini impurity
- **Training Process:**
  - Use the neural network's predictions as the target output.
  - Train two decision trees:
    - **With Feature Z:** Combine features $X_1, X_2$ and $Z$ into a single input matrix
    - **Without Feature Z:** Use only the original features $X_1$ and $X_2$
- **Output:**
  - Two decision trees: one incorporating the derived feature and one without it, both trained to approximate the neural network's decision boundary.

General instructions for reproducing the model training process for any two-class classification problem:

1. Generate or load the dataset: Ensure the dataset has two features which can be represented by $X_1$ and $X_2$, and two class labels (e.g., 0 and 1).
2. Train the Neural Network: Use any Neural Network model with a specified configuration for training to make sure it learns the boundary.
3. Define the derived feature: Analyse the geometry of the decision boundary of the underlying data by visualising it with the help of a scatter plot and define Z accordingly. The feature Z should encapsulate the underlying pattern that distinguishes the two classes.
4. Train Decision Trees: Train a decision tree using the combined features $X_1, X_2, Z$ and another decision tree using the original features $X_1, X_2$
5. Evaluate and Compare Models: Use evaluation metrics like accuracy, precision, recall, F1-score, and ROC-AUC to compare the decision trees with the neural network and other models, focusing on interpretability and accuracy.

## 4.2 Results for data with Linear Decision Boundary

### 4.2.1 Neural Network-Based Linear Decision Boundary Analysis

In this section, we evaluate the performance of a neural network trained on data with a linear decision boundary which is parallel to the Y-Axis. A synthetic dataset was generated where each point $(X_1, X_2)$ is uniformly sampled from the range $[-1, 1]$. The labels for the data points were assigned based on the condition $X_1 > 0$, creating a linear decision boundary. The neural network used for training had a single hidden layer with 20 neurons and was trained using the $MLPClassifier$ from scikit-learn.

The decision boundary formed by the neural network is depicted in **Figure 1** below. The neural network successfully learns the linear boundary, as shown by the clear separation of the two classes (Red and Blue).

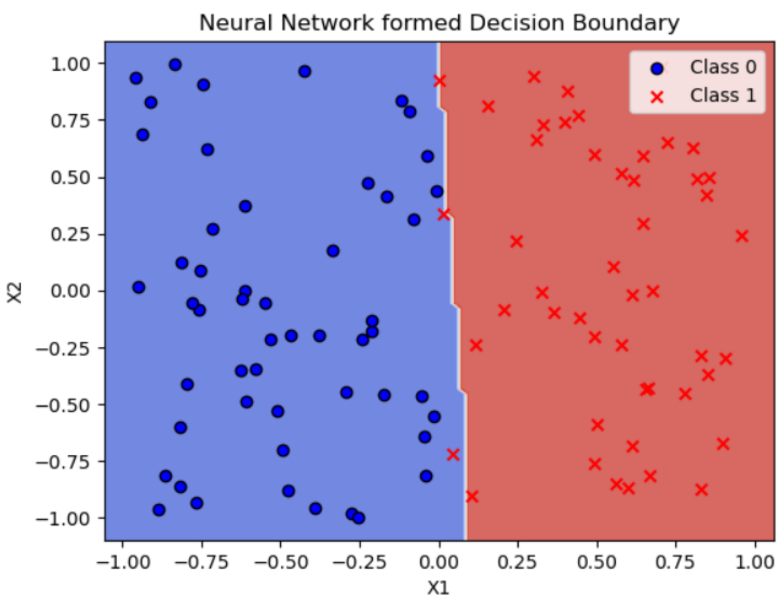

**Figure 4.1:** Linear Decision Boundary formed by the Neural Network

### *4.2.2 Decision Tree Approximation with Derived Feature Z*

To further analyse the interpretability of the neural network, we introduced a derived feature Z, defined as a linear combination of $X_1$ and $X_2$ specifically $Z = X_1 + X_2$ but since the decision boundary is parallel to the Y-Axis, we do not need $X_2$ to capture the essence of the underlying data since that will solely depend on $X_1$. This derived feature aims to simplify the decision-making process by capturing the intrinsic linear relationship present in the data.

A decision tree was trained using the neural network's predictions along with the combined features $(X_1, X_2, Z)$. The resulting decision tree, as shown in **Figure 2**, demonstrates a clear, interpretable rule based on the derived feature Z. The tree correctly classifies the data points with an accuracy of 100%, leveraging the linearity of Z. This is evident in the decision node that checks if $Z \leq 0.075$, which separates the classes perfectly.

Additionally, a second decision tree was trained without the derived feature Z, using only the original features $(X_1, X_2)$. As shown in **Figure 3**, the tree struggles to achieve the same level of interpretability and accuracy, reflecting the importance of the derived feature in simplifying the decision boundary.

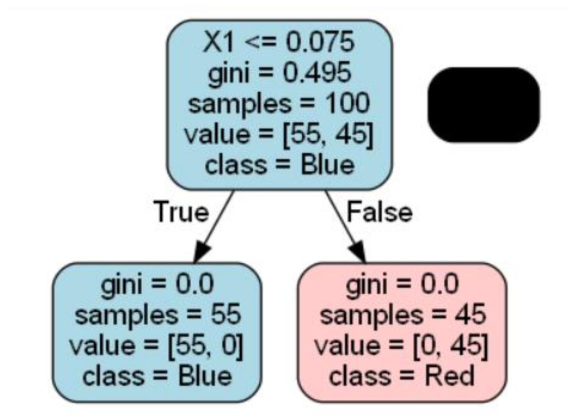

**Figure 4.2:** Decision Tree for Data with Linear Decision Boundary with Feature Z (Using Neural Networks' Predictions)

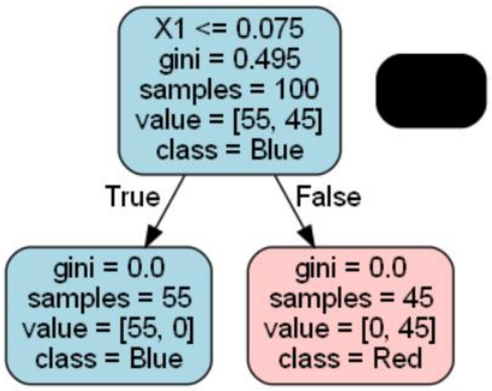

**Figure 4.3:** Decision Tree for Data with Linear Decision Boundary without Feature Z (Using Neural Networks' Predictions)

**Mathematical Condition and Its Evaluation:**

The decision tree with the derived feature Z generates a simple mathematical condition that represents the decision boundary learned by the neural network. In this case, the condition formed is:

$$IF\ X_1 \leq\ 0.075\ ?\ Blue\ :\ Red$$

This condition is tested against the dataset to evaluate its accuracy. The accuracy is calculated by comparing the predictions made using this condition with the predictions of the neural network. The result shows an accuracy of 100%, confirming that the derived equation effectively captures the decision boundary created by the neural network.

**Evaluation Metrics Summary:**

The models were evaluated using various metrics to assess their performance on the linear decision boundary dataset:

- **Accuracy**: The neural network achieved an accuracy of 97%, while the decision tree with the derived feature Z achieved 100%, and without Z also achieved 100%. The Linear SVM obtained an accuracy of 98%.
- **Precision, Recall, and F1-Score**: The neural network and decision tree with Z both achieved high scores across these metrics, demonstrating their effectiveness in classification.
- **ROC-AUC**: Both models achieved a ROC-AUC of 1.0, indicating their excellent performance in distinguishing between the classes.

## 4.3 Results for data with Oblique Decision Boundary

### 4.3.1 Neural Network-Based Oblique Decision Boundary Analysis

In this section, we evaluate the performance of a neural network trained on data with an oblique decision boundary. A synthetic dataset was generated where each point $(X_1, X_2)$ is uniformly sampled from the range $[-1, 1]$. The labels for the data points were assigned based on the condition $X_1 + X_2 > 0$, creating an oblique decision boundary. The neural network used for training had a single hidden layer with 20 neurons and was trained using the $MLPClassifier$ from scikit-learn.

The decision boundary formed by the neural network is depicted in **Figure 1** below. The neural network successfully learns the oblique boundary, as shown by the clear separation of the two classes (Red and Blue).

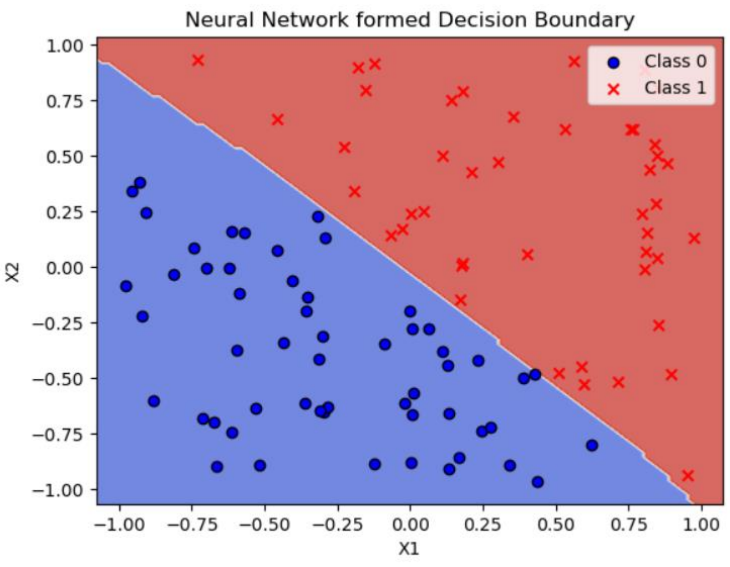

**Figure 4.4:** Oblique Decision Boundary formed by the Neural Network

### *4.3.2 Decision Tree Approximation with Derived Feature*

To further analyse the interpretability of the neural network, we introduced a derived feature Z, defined as a linear combination of $X_1$ and $X_2$ specifically $Z = X_1 + X_2$. This derived feature aims to simplify the decision-making process by capturing the underlying relationship present in the data.

A decision tree was trained using the neural network's predictions along with the combined features $(X_1, X_2, Z)$. The resulting decision tree, as shown in **Figure 2**, demonstrates a clear, interpretable rule based on the derived feature Z. The tree correctly classifies the data points with an accuracy of 100%, leveraging the linearity of Z. This is evident in the decision node that checks if $Z \leq -0.020$, which separates the classes perfectly.

Additionally, a second decision tree was trained without the derived feature Z, using only the original features $(X_1, X_2)$. As shown in **Figure 3**, the tree struggles to achieve the same level of interpretability and accuracy, reflecting the importance of the derived feature in simplifying the decision boundary.

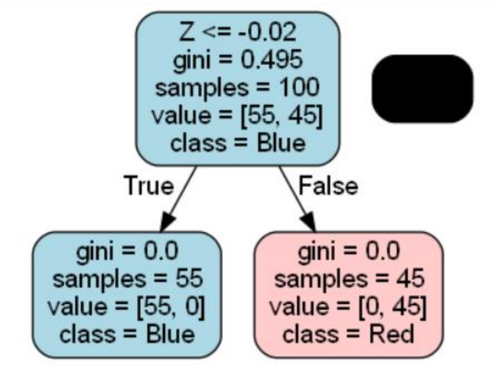

**Figure 4.5:** Decision Tree for Data with Oblique Decision Boundary with Feature Z (Using Neural Networks' Predictions)

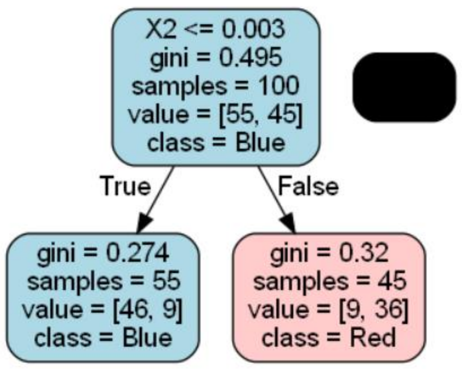

**Figure 4.6:** Decision Tree for Data with Oblique Decision Boundary without Feature Z (Using Neural Networks' Predictions)

**Mathematical Condition and Its Evaluation:**

The decision tree with the derived feature Z generates a simple mathematical condition that represents the decision boundary learned by the neural network. In this case, the condition formed is:

$$IF\ X_1 + X_2 \leq\ -0.020\ ?\ Blue\ :\ Red$$

This condition is tested against the dataset to evaluate its accuracy. The accuracy is calculated by comparing the predictions made using this condition with the predictions of the neural network. The result shows an accuracy of 100%, confirming that the derived equation effectively captures the decision boundary created by the neural network.

**Evaluation Metrics Summary:**

The models were evaluated using various metrics to assess their performance on the linear decision boundary dataset:

- **Accuracy**: The neural network achieved an accuracy of 100%, while the decision tree with the derived feature Z achieved 100%, and without Z, it achieved 82%. The Linear SVM also obtained an accuracy of 100%.
- **Precision, Recall, and F1-Score**: The neural network and decision tree with Z both achieved high scores across these metrics, demonstrating their effectiveness in classification.
- **ROC-AUC**: Both models achieved a ROC-AUC of 1.0, indicating excellent performance in distinguishing between the classes.

## 4.4 Results for data with Circular Decision Boundary

### 4.4.1 Neural Network-Based Circular Decision Boundary Analysis

In this section, we evaluate the performance of a neural network trained on data with a circular decision boundary. A synthetic dataset was generated where each point $(X_1, X_2)$ is uniformly sampled from the range $[-1, 1]$. The labels for the data points were assigned based on the condition $X_1^2 + X_2^2 > 0.5$, creating a circular decision boundary. The neural network used for training had a single hidden layer with 20 neurons and was trained using the $MLPClassifier$ from scikit-learn.

The decision boundary formed by the neural network is depicted in **Figure 1**. The neural network successfully learns the circular boundary, evident from the separation of the two classes (Red and Blue) along the circular region.

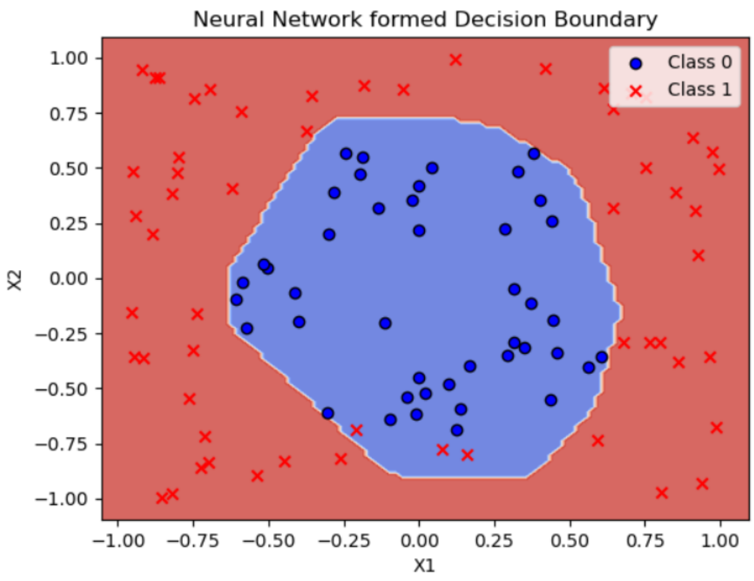

**Figure 4.7:** Circular Decision Boundary formed by the Neural Network

### *4.4.2 Decision Tree Approximation with Derived Feature Z*

To further analyse the interpretability of the neural network, we introduced a derived feature Z, defined as sum of squares of $X_1$ and $X_2$ specifically $Z = X_1^2 + X_2^2$. This derived feature aims to simplify the decision-making process by capturing the intrinsic circular relationship present in the data.

A decision tree was trained using the neural network's predictions along with the combined features $(X_1, X_2, Z)$. The resulting decision tree, as shown in **Figure 2**, demonstrates a clear, interpretable rule based on the derived feature Z. The tree correctly classifies the data points with an accuracy of 98%, leveraging the circular nature of Z. This is evident in the decision node that checks if $Z \leq 0.516$, the classes are nearly separated perfectly.

Additionally, a second decision tree was trained without the derived feature Z, using only the original features $(X_1, X_2)$. As shown in **Figure 3**, the tree struggles to achieve the same level of interpretability and accuracy, reflecting the importance of the derived feature in simplifying the decision boundary.

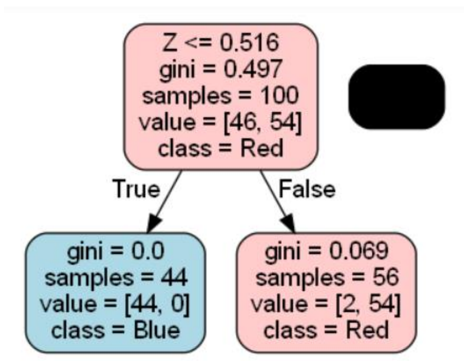

**Figure 4.8:** Decision Tree for Data with Circular Decision Boundary with Feature Z (Using Neural Networks' Predictions)

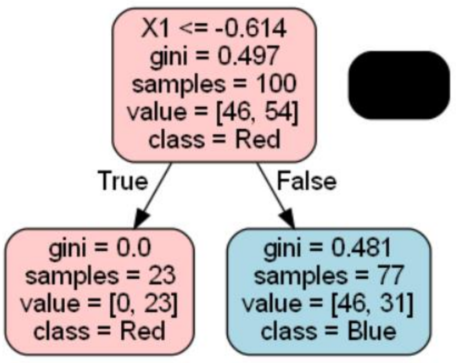

**Figure 4.9:** Decision Tree for Data with Circular Decision Boundary without Feature Z (Using Neural Networks' Predictions)

**Mathematical Condition and Its Evaluation:**

The decision tree with the derived feature Z generates a simple mathematical condition that represents the decision boundary learned by the neural network. In this case, the condition formed is:

$$IF\ X_1^2 + X_2^2 \leq\ 0.516\ ?\ Blue : Red$$

This condition is tested against the dataset to evaluate its accuracy. The accuracy is calculated by comparing the predictions made using this condition with the predictions of the neural network. The result shows an accuracy of 98%, confirming that the derived equation effectively captures the decision boundary created by the neural network.

**Evaluation Metrics Summary:**

The models were evaluated using various metrics to assess their performance on the circular decision boundary dataset:

- **Accuracy**: The neural network achieved an accuracy of 97%, while the decision tree with the derived feature Z achieved 98%, and without Z, it achieved 69%. Logistic Regression with Polynomial Features obtained an accuracy of 97%.
- **Precision, Recall, and F1-Score**: The neural network and decision tree with Z both achieved high scores across these metrics, demonstrating their effectiveness in classification.
- **ROC-AUC**: Neural Network achieved an ROC-AUC of 0.99 whereas Decision Tree achieved an ROC-AUC of 0.98, reflecting good performance in distinguishing between the classes.

## 4.5 Results for data with Elliptical Decision Boundary

### 4.5.1 Neural Network-Based Elliptical Decision Boundary Analysis

In this section, we evaluate the performance of a neural network trained on data with a elliptical decision boundary. A synthetic dataset was generated where each point $(X_1, X_2)$ is uniformly sampled from the range $[-1, 1]$. The classification labels were assigned based on the condition $\frac{X_1^2}{a^2} + \frac{X_2^2}{b^2} > 1$, where $a = 0.7$ and $b = 0.5$ represent the semi-major and semi-minor axes, respectively, forming an elliptical decision boundary. The neural network used for training had a single hidden layer with 20 neurons and was trained using the $MLPClassifier$ from scikit-learn.

The decision boundary formed by the neural network is illustrated in **Figure 1**. The neural network learns the elliptical boundary, evident from the separation of the two classes (Red and Blue) along the elliptical region.

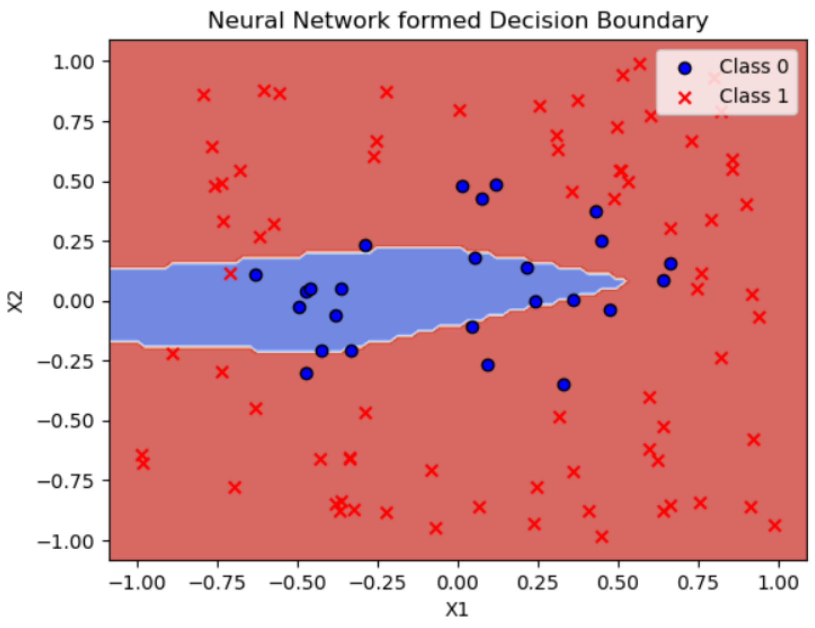

**Figure 4.10:** Elliptical Decision Boundary formed by the Neural Network

### *4.5.2 Decision Tree Approximation with Derived Feature Z*

To further analyse the interpretability of the neural network, we introduced a derived feature Z, defined as $Z = \frac{X_1^2}{a^2} + \frac{X_2^2}{b^2}$. This derived feature aims to simplify the decision-making process by capturing the intrinsic elliptical relationship present in the data.

A decision tree was trained using the neural network's predictions along with the combined features $(X_1, X_2, Z)$. The resulting decision tree, as shown in **Figure 2**, demonstrates a clear, interpretable rule based on the derived feature Z. The tree classifies the data points with an accuracy of 92%, leveraging the elliptical nature of Z. This is evident in the decision node that checks if $Z \leq 0.601$, which nearly separates the two classes, emphasizing the value of the derived feature in simplifying the model's interpretation.

Additionally, a second decision tree was trained without the derived feature Z, using only the original features $(X_1, X_2)$. As shown in **Figure 3**, the tree exhibits a lower classification accuracy, highlighting the importance of the derived feature in capturing the elliptical decision boundary.

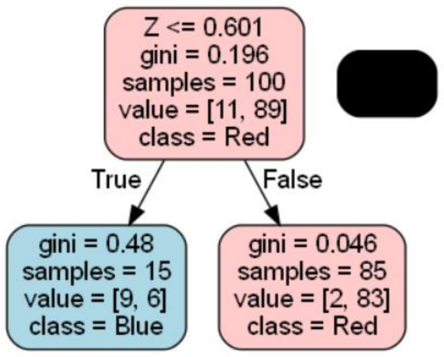

**Figure 4.11:** Decision Tree for Data with Elliptical Decision Boundary with Feature Z (Using Neural Networks' Predictions)

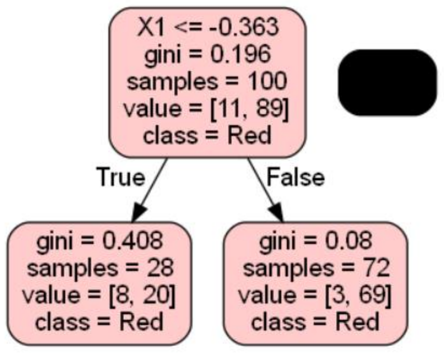

**Figure 4.12:** Decision Tree for Data with Elliptical Decision Boundary without Feature Z (Using Neural Networks' Predictions)

**Mathematical Condition and Its Evaluation:**

The decision tree with the derived feature Z generates a simple mathematical condition that represents the decision boundary learned by the neural network. In this case, the condition formed is:

$$IF\ \frac{X_1^2}{a^2}+\frac{X_2^2}{b^2}\leq\ 0.601\ ?\,Blue\ :\ Red$$

This condition is tested against the dataset to evaluate its accuracy. The accuracy is calculated by comparing the predictions made using this condition with the predictions of the neural network. The result shows an accuracy of 92%, indicating that the derived equation effectively captures the decision boundary created by the neural network.

**Evaluation Metrics Summary:**

The models were evaluated using various metrics to assess their performance on the elliptical decision boundary dataset:

- **Accuracy**: The neural network achieved an accuracy of 84%, while the decision tree with the derived feature Z achieved 92%, and without Z, it achieved 89%.
- **Precision, Recall, and F1-Score**: The neural network and decision tree with Z both achieved high scores across these metrics, demonstrating their effectiveness in classification.
- **ROC-AUC**: The neural network obtained an ROC-AUC of 0.95, while the decision tree with Z achieved 0.88, reflecting good performance in distinguishing between the classes.

## 4.6 Results for data with Square-shaped Decision Boundary

### 4.6.1 Neural Network-Based Square-shaped Decision Boundary Analysis

In this section, we evaluate the performance of a neural network trained on data with a square-shaped decision boundary. A synthetic dataset was generated where each point $(X_1, X_2)$ is uniformly sampled from the range $[-1, 1]$. The classification labels were assigned based on the condition $\max\left(abs(X_1), abs(X_2)\right) > 0.5$, forming a square-shaped decision boundary. The neural network used for training had a single hidden layer with 20 neurons and was trained using the $MLPClassifier$ from scikit-learn.

The decision boundary formed by the neural network is illustrated in **Figure 1**. The neural network learns the square-shaped boundary, evident from the separation of the two classes (Red and Blue).

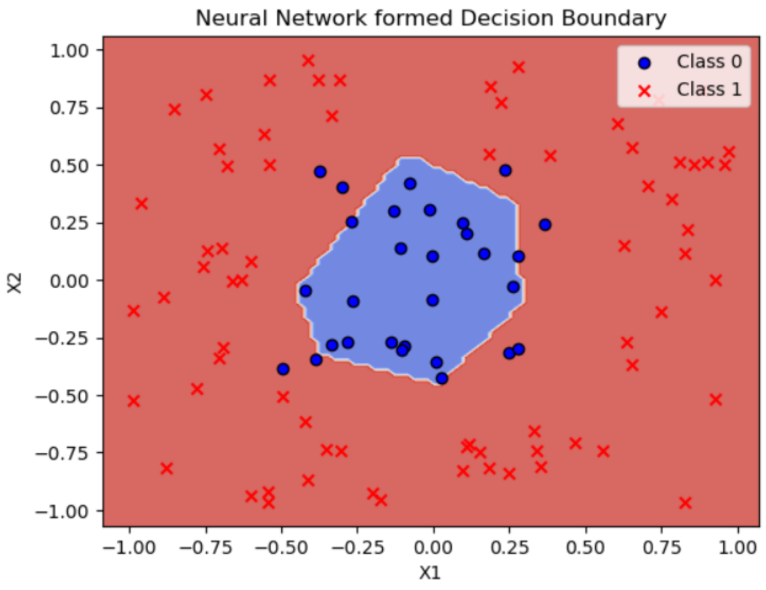

**Figure 4.13:** Square-shaped Decision Boundary formed by the Neural Network

### 4.6.2 Decision Tree Approximation with Derived Feature Z

To further analyse the interpretability of the neural network, we introduced a derived feature Z, defined as $Z = \max\left(abs(X_1), abs(X_2)\right)$. This derived feature aims to simplify the decision-making process by capturing the intrinsic square-shaped relationship present in the data.

A decision tree was trained using the neural network's predictions along with the combined features $(X_1, X_2, Z)$. The resulting decision tree, as shown in **Figure 2**, demonstrates a clear, interpretable rule based on the derived feature Z. The tree effectively classifies the data points with an accuracy of 95%, leveraging the square-shaped nature of Z. This is evident in the decision node that checks if $Z \leq 0.448$, which neatly separates the two classes.

Additionally, a second decision tree was trained without the derived feature Z, using only the original features $(X_1, X_2)$. The tree's structure, shown in Figure 9, is more complex and less interpretable, reflecting the importance of the derived feature in accurately capturing the decision boundary's shape.

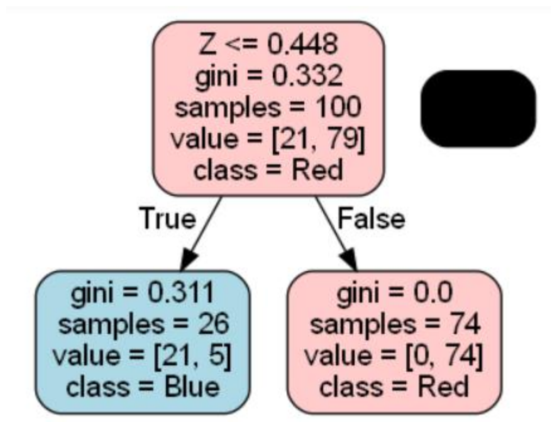

**Figure 4.14:** Decision Tree for Data with Square-shaped Decision Boundary with Feature Z (Using Neural Networks' Predictions)

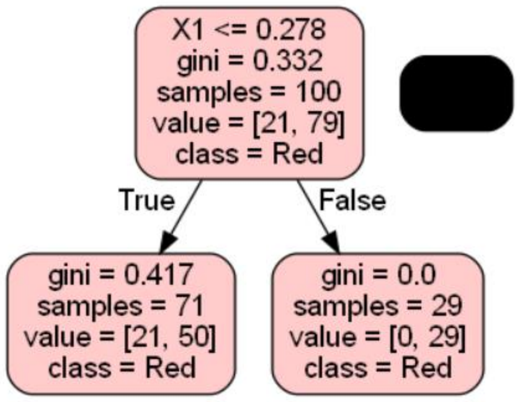

**Figure 4.15:** Decision Tree for Data with Square-shaped Decision Boundary without Feature Z (Using Neural Networks' Predictions)

**Mathematical Condition and Its Evaluation:**

The decision tree with the derived feature Z generates a simple mathematical condition that represents the decision boundary learned by the neural network. In this case, the condition formed is:

$$IF \max\left(abs(X_1), abs(X_2)\right) \leq \ 0.448\ ?\, Blue : Red$$

This condition is tested against the dataset to evaluate its accuracy. The accuracy is calculated by comparing the predictions made using this condition with the predictions of the neural network. The result shows an accuracy of 97%, indicating that the derived equation effectively captures the decision boundary created by the neural network.

**Evaluation Metrics Summary:**

The models were evaluated using various metrics to assess their performance on the square-shaped decision boundary dataset:

- **Accuracy**: The neural network achieved an accuracy of 92%, while the decision tree with the derived feature Z achieved 95%, and without Z, it achieved 79%.

- **Precision, Recall, and F1-Score**: The neural network and decision tree with Z both achieved high scores across these metrics, confirming their effectiveness in classification.
- **ROC-AUC**: The neural network obtained a ROC-AUC of 1, while the decision tree with Z achieved 0.97, reflecting strong performance in distinguishing between the classes.

## 4.7 Summary of Experimental Results

The experiments conducted in this study effectively demonstrated the capabilities of neural networks after integrating it with decision trees to classify data for different decision boundaries like linear, oblique, circular, elliptical, and square. Neural networks achieved consistently high accuracy, with results varying between 84% and 100% across different types of boundaries. Decision trees, when enhanced with appropriate derived features along with neural networks' predictions, demonstrated comparable or superior performance, achieving up to 100% accuracy while also offering better interpretability by mimicking the neural network. However, without these derived features, decision tree accuracy decreased significantly, highlighting the critical role of feature engineering. Comparisons with other models, such as Linear SVM and Logistic Regression with polynomial features, supported these findings, confirming that the careful selection of features enhances both accuracy and model transparency.

# Chapter 5: Discussions

## 5.1 Analysis of Results

This research demonstrates that decision trees, when integrated with a derived feature that captures the pattern in the underlying data, can adeptly follow the classification patterns of neural networks, addressing the challenge of the "black box" nature of such networks by making their decisions more interpretable. Throughout the experiments, neural networks consistently delivered high accuracy across various types of decision boundaries (linear, oblique, circular, elliptical, and square). Remarkably, decision trees, guided by the engineered feature, often reached comparable or even higher levels of accuracy, effectively mimicking the neural networks' classification strategies. These outcomes highlight the transformative potential of feature engineering to reconcile the demands of model interpretability with high performance, thereby enhancing our ability to trust and comprehend the decisions made by sophisticated models like neural networks.

## 5.2 Advantages of using Derived Feature in Decision Trees

The integration of the derived feature was instrumental in empowering decision trees to closely replicate the decision-making processes of neural networks. By implementing this new feature, decision trees were able to perform clear and precise splits that mirrored the decision boundaries established by the neural networks. This enhancement streamlined the structure of the trees, minimised their depth, and rendered the decision rules more transparent and interpretable. The experimental results underscore that employing derived features is a potent solution to the interpretability challenges often posed by neural networks, paving the way for a deeper understanding of decision-making processes without sacrificing accuracy. This approach enables decision trees to act as surrogate models, shedding light on the sophisticated decision logic embedded within neural networks.

## 5.3 Interpretability vs. Accuracy: Balancing Trade-Offs

The study also highlights how the use of derived features that captures the underlying data like Z can mitigate the often-competing demands of interpretability and accuracy in machine learning models. While neural networks deliver exceptional predictive performance, their "black box" nature can hinder their application in areas where understanding model rationale is paramount. Augmenting decision trees with a feature that captures the nature of the underlying data not only maintains high accuracy but also enhances interpretability, providing clear insights into how decisions are formulated.

## 5.4 Implications for Real-World Applications

The implications of this research are significant for real-world environments where both high accuracy and transparency are essential. This study demonstrates that decision trees, supplemented with derived features, can effectively mimic neural networks' decision-making strategies, thus addressing the pervasive "black box" issue in complex models like neural networks. This finding is especially valuable in high-stakes fields such as healthcare, law, and finance, where understanding the rationale behind AI decisions is crucial.

## 5.5 Comparison with Existing Interpretability Techniques

Our method of incorporating features that give insights about the underlying data for enhancing interpretability offers an innovative alternative to established techniques such as LIME and SHAP, which analyse model predictions after the fact. In contrast to these post-hoc explanation methods, our approach facilitates inherent interpretability by creating a surrogate decision tree that emulates the behaviours of the neural network. This method not only integrates interpretability into the very fabric of the model but also promises a more straightforward and possibly more reliable means of understanding the rationale behind neural network decisions.

## 5.6 Limitations of the Current Study

While the study demonstrates the potential of using derived features to enhance interpretability, several limitations exist. The experiments were conducted on synthetic datasets, which may not capture all the complexities and nuances of real-world data. Additionally, the study focused on binary classification tasks; the approach may require further validation and adjustments to handle multi-class or multi-dimensional data effectively. Future research should explore the scalability of this method and its applicability to other types of machine learning models and datasets.

# Chapter 6: Conclusion

## *6.1 Summary of Key Findings*

1. **Improved Interpretability through Integration:** The research demonstrates that decision trees can effectively mimic the decision boundaries learned by neural networks when augmented with a strategically derived feature that captures the nature of the underlying data. This integration offers a more interpretable alternative to the black-box nature of neural networks, bridging the gap between high accuracy and transparency in machine learning models.
2. **Effectiveness across Diverse Decision Boundaries:** The approach was tested on various datasets representing different decision boundaries (linear, oblique, circular, elliptical, and square). In each case, the derived feature enabled decision trees to approximate the neural network's classification strategy, maintaining high interpretability while achieving comparable accuracy levels.
3. **Potential for Broader Applications:** The findings suggest that this method can be applied to any complex machine learning model beyond neural networks, enhancing interpretability without sacrificing performance. This is particularly valuable for domains requiring high-stakes decision-making, such as healthcare, finance, and legal contexts.
4. **Balancing Complexity and Simplicity:** The research highlights that the integration of decision trees with neural networks balances the complexity of neural networks with the simplicity of decision trees. This dual approach allows for both a deep understanding of complex patterns and the ability to provide straightforward, single boolean equation based explanation, which is crucial for gaining trust in AI systems.

## *6.2 Future Research Directions*

1. **Application to Other Models**: Expand this augmented approach to other such black-box models like deep neural networks and ensemble methods, exploring different types of derived features to enhance interpretability.
2. **Automated Feature Discovery**: Develop methods to automate the derivation of new features that best capture decision boundaries, reducing manual effort and improving scalability.
3. **Integration with Domain Knowledge**: Tailor the approach by incorporating domain-specific insights, enhancing its relevance and applicability in fields like medicine, finance, and law.

## *6.3 Concluding Remarks*

This research contributes to the field of explainable AI by demonstrating how decision trees can effectively mimic the decision-making processes of neural networks, thereby enhancing model interpretability without compromising accuracy. By leveraging a derived feature to capture essential patterns within the data, the study bridges the gap between the high performance of neural networks and the transparency required for practical, real-world applications. Future research should focus on refining this approach to accommodate various complex models and integrating it with domain-specific knowledge to further enhance its applicability and utility. This work paves the way for more trustworthy and transparent AI systems, aligning with ethical standards and increasing acceptance in high-stakes domains like healthcare, finance, and law.

## References

Aytekin, C. (2022) 'Neural Networks are Decision Trees'. Available at: http://arxiv.org/abs/2210.05189.

Blanco-Justicia, A. *et al.* (2020) 'Machine learning explainability via microaggregation and shallow decision trees', *Knowledge-Based Systems*, 194, p. 105532. Available at: https://doi.org/10.1016/j.knosys.2020.105532.

Chakraborty, M. (2024) 'Explainable Neural Networks: Achieving Interpretability in Neural Models', *Archives of Computational Methods in Engineering* [Preprint]. Available at: https://doi.org/10.1007/s11831-024-10089-4.

Cortés-Ferre, L. *et al.* (2023) 'Deep Learning Applied to Intracranial Hemorrhage Detection', *Journal of Imaging*, 9(2), p. 37. Available at: https://doi.org/10.3390/jimaging9020037.

Craven, M.W. and Shavlik, J.W. (1997) *Understanding Time-Series Networks: A Case Study in Rule Extraction, International Journal of Neural Systems special issue on Noisy Time Series*.

Cybenko, G. and Cybenkot, G. (1989) 'Approximation by superpositions of a sigmoidal function Approximation by Superpositions of a Sigmoidal Function*', 2(4). Available at: https://doi.org/10.1007/BF02551274ï.

Doshi-Velez, F. and Kim, B. (no date) *Towards A Rigorous Science of Interpretable Machine Learning*.

Duede, E. (2023) 'Deep Learning Opacity in Scientific Discovery', *Philosophy of Science*, 90(5), pp. 1089–1099. Available at: https://doi.org/10.1017/psa.2023.8.

Ferigo, A., Custode, L.L. and Iacca, G. (2023) 'Quality–diversity optimization of decision trees for interpretable reinforcement learning', *Neural Computing and Applications* [Preprint]. Available at: https://doi.org/10.1007/s00521-023-09124-5.

Friedman, J.H. and Popescu, B.E. (2008) 'Predictive learning via rule ensembles', *Annals of Applied Statistics*, 2(3), pp. 916–954. Available at: https://doi.org/10.1214/07-AOAS148.

Frosst, N. and Hinton, G. (2017) 'Distilling a Neural Network Into a Soft Decision Tree'. Available at: http://arxiv.org/abs/1711.09784.

Gunning, D. *et al.* (2019) 'XAI—Explainable artificial intelligence', *Science Robotics*, 4(37). Available at: https://doi.org/10.1126/scirobotics.aay7120.

Gupta, A., Park, S. and Lam, S.M. (1999) *Concise Papers_____ Generalized Analytic Rule Extraction for Feedforward Neural Networks, iEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*.

Heaton, J. (2018) 'Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning', *Genetic Programming and Evolvable Machines*, 19(1–2), pp. 305–307. Available at: https://doi.org/10.1007/s10710-017-9314-z.

Ismail Fawaz, H. *et al.* (2019) 'Deep learning for time series classification: a review', *Data Mining and Knowledge Discovery*, 33(4), pp. 917–963. Available at: https://doi.org/10.1007/s10618-019-00619-1.

Krishnan, R., Sivakumar, G. and Bhattacharya, P. (no date) *Extracting decision trees from trained neural networks*.

Lundberg, S.M., Allen, P.G. and Lee, S.-I. (2017) *A Unified Approach to Interpreting Model Predictions*. Available at: https://github.com/slundberg/shap.

Lundberg, S.M., Erion, G.G. and Lee, S.-I. (2018) 'Consistent Individualized Feature Attribution for Tree Ensembles'. Available at: http://arxiv.org/abs/1802.03888.

Minh, D. *et al.* (2022) 'Explainable artificial intelligence: a comprehensive review', *Artificial Intelligence Review*, 55(5), pp. 3503–3568. Available at: https://doi.org/10.1007/s10462-021-10088-y.

Ribeiro, M.T., Singh, S. and Guestrin, C. (2016) '"Why should i trust you?" Explaining the predictions of any classifier', in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, pp. 1135–1144. Available at: https://doi.org/10.1145/2939672.2939778.

Rudin, C. (2019) 'Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead', *Nature Machine Intelligence*, 1(5), pp. 206–215. Available at: https://doi.org/10.1038/s42256-019-0048-x.

Saraswat, D. *et al.* (2022) 'Explainable AI for Healthcare 5.0: Opportunities and Challenges', *IEEE Access*. Institute of Electrical and Electronics Engineers Inc., pp. 84486–84517. Available at: https://doi.org/10.1109/ACCESS.2022.3197671.

Selvaraju, R.R. *et al.* (2017) *Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization*. Available at: http://gradcam.cloudcv.org.

Štrumbelj, E. and Kononenko, I. (2014) 'Explaining prediction models and individual predictions with feature contributions', *Knowledge and Information Systems*, 41(3), pp. 647–665. Available at: https://doi.org/10.1007/s10115-013-0679-x.

Zhang, Q. *et al.* (no date) *Interpreting CNNs via Decision Trees*.

Nagyfi, R., 2021. *The Differences Between Artificial and Biological Neural Networks.* [Online]
Available at: https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7
[Accessed 05 09 2021].