Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Predicting Creditworthiness with Artificial Neural Networks and XGBoost: An Explainability Approach using SHAP and ALE

Research Paper

Name of the Study Program

Finanzmathematik, Aktuarwissenschaften und Risikomanagement

Faculty 4

submitted by

Duc Tung Bui (s0575652)

Raika Hoorsun (s0559353)

Date:

Berlin, February 09, 2025

Examiner: Dr. Alla Petukhina

Contents

1 Introduction

Creditworthiness refers to an individual's or entity's ability to meet financial obligations and repay borrowed funds. It plays a crucial role in financial decision-making, influencing loan approvals, interest rates, and overall access to credit. Traditionally, creditworthiness has been assessed using financial indicators such as credit scores, debt-to-income ratios, and past repayment behavior. However, with the rise of Artificial Intelligence (AI) and Machine Learning (ML), predictive models are increasingly used to analyze complex patterns in financial data, offering more accurate and dynamic assessments.

This study is divided into several chapters, which allow for a systematic investigation of the problem, the theoretical foundations, the applied methods, and the results achieved by utilizing the German Credit Data set. Chapter 2 provides a comprehensive overview of the theoretical background, focusing on the significance of credibility in decision-making processes. It also explores the growing relevance of artificial intelligence (AI), highlighting how AI technologies are increasingly pivotal in making informed, data-driven decisions. Chapters 3 and 4 delve into the methodology of the machine learning models used in this research, specifically Artificial Neural Networks (ANN) and eXtreme Gradient Boosting (XGBoost). These chapters describe the architecture, training procedures, and key parameters of the models, as well as their application in solving the research problem. In Chapter 5, the paper outlines the methodology behind the post hoc explainability techniques applied to the models, particularly SHAP (SHapley Additive exPlanations) and ALE (Accumulated Local Effects). This chapter explains how these methods were used to interpret and explain the predictions made by the machine learning models, providing transparency and understanding of model behavior. Chapter 6 presents the results of the model performance and the post hoc analysis. The findings are detailed and analyzed, offering insights into the accuracy, interpretability, and overall effectiveness of the applied models. Chapter 7 discusses the implications of the results, comparing them with existing literature and theoretical expectations. It addresses potential limitations and suggests areas for further research, providing a deeper understanding of the research findings. Finally, Chapter 8 concludes the paper by summarizing the key findings and offering an outlook on future developments in the field. It reflects on the significance of the research, potential real-world applications, and avenues for advancing both machine learning and explainability methods.

2 Theoretical Background

Creditworthiness is a fundamental concept in finance, determining an individual's or entity's ability to meet financial obligations. Traditionally, it has been assessed using financial indicators such as credit scores, debt-to-income ratios, and repayment history. While these methods provide useful insights, they often fail to capture complex interactions between various financial and non-financial factors, leading to potential inaccuracies in credit risk assessment[1]. With advancements in Artificial Intelligence (AI) and Machine Learning (ML), more sophisticated models are being utilized to enhance creditworthiness prediction. Techniques such as Artificial Neural Networks (ANN) and XGBoost can analyze large datasets, detect intricate patterns, and improve predictive accuracy. However, despite their effectiveness, these models often operate as "black boxes," making it difficult to understand how decisions are made. To address this challenge, post hoc explainability methods such as SHAP (SHapley Additive Explanations) and ALE (Accumulated Local Effects) are employed. These techniques help interpret model outputs by identifying key features that influence predictions, enhancing transparency and trust in AI-driven credit scoring. By combining predictive accuracy with explainability, AI-based approaches offer a more robust framework for assessing creditworthiness while ensuring accountability in financial decision-making[2].

2.1 Relevance of AI for predicting creditworthiness

The use of AI and ML in predicting creditworthiness has become increasingly important in the financial sector. Traditional credit scoring models, while effective, often rely on a limited set of criteria and may overlook important patterns within the data. In contrast, AI and ML algorithms can analyze vast amounts of structured and unstructured data, capturing complex relationships and trends that may be missed by conventional methods. These models can significantly enhance the accuracy and efficiency of creditworthiness assessments, leading to more informed lending decisions and better risk management.

One of the key advantages of AI in this context is its ability to continuously improve. As new data becomes available, AI models can be retrained to adapt to changing economic conditions, customer behaviors, and emerging risks, ensuring that credit assessments remain relevant and up-to-date. Furthermore, machine learning techniques, such as deep learning, can identify intricate patterns in the data that help in distinguishing between high and low credit risks, making credit scoring systems more robust and predictive. However, the application of AI and ML in creditworthiness prediction comes with challenges, particularly regarding the "black box" problem. Many ML models, while highly accurate, operate in ways that are not easily interpretable, making it difficult for stakeholders to understand how decisions are made. This lack of transparency can be problematic, especially in industries like finance where decisions have significant implications for individuals and organizations[3]. To address this issue, several explainability methods have been developed, such as SHAP (SHapley Additive exPlanations) and ALE (Accumulated Local Effects). These techniques aim to provide insight into the inner workings of complex

[1] Turkson, Baagyere, and Wenya 2016, p. 81, 82.
[2] Chang et al. 2024, p. 2, 3.
[3] Orlova 2021, p. 1, 2.

models by offering interpretable explanations for individual predictions. By shedding light on the factors that influence a model's decision-making process, these explainability methods help build trust and accountability in AI-driven systems, making them more suitable for high-stakes applications like creditworthiness prediction[4].

In summary, AI and machine learning are transforming the way creditworthiness is assessed by leveraging advanced techniques to improve accuracy and adaptability. However, the challenge of model interpretability is being addressed through explainability methods, which ensure that these powerful tools can be applied transparently and responsibly in financial decision-making.

[4]Segun, Yemi-Peters, and Adewumi 2024, p. 15, 16.

3 Artificial Neural Networks

Artificial Neural Networks are powerful tools for modeling complex nonlinear relationships between inputs and outputs or for recognizing patterns in data[5]. They consist of interconnected nodes, called neurons, organized into three main layers: Input layer, hidden layer and output layer. In this configuration, the input nodes do not perform computations but are used to feed information into the network[6].

These models are characterized by their structure, the activation function used and their learning algorithm. A network as in Fig. 1 is called a feedforward network because the connections between successive layers are unidirectional and they move exclusively from the input to the output layer without going through backward connections or feedback loops[7]. The feedforward architecture comprises one or more non-linear hidden layers and a linear output layer.

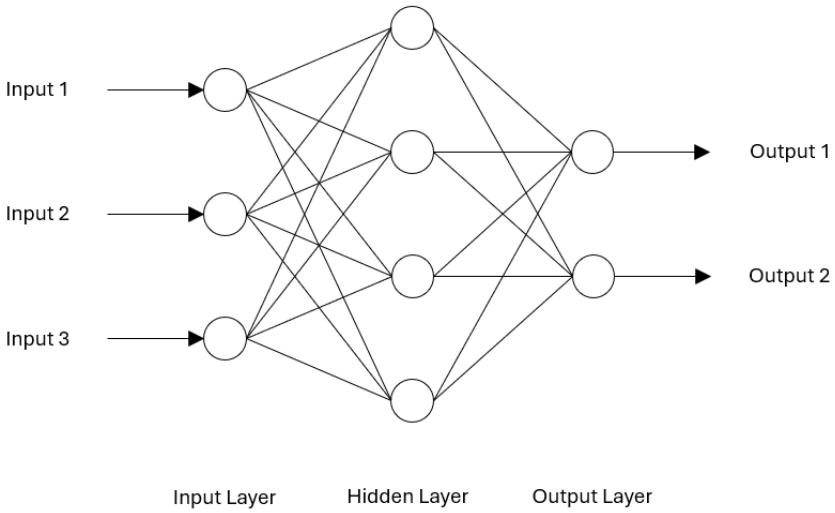

Figure 1: Topology of an artificial neural network with 3 inputs and 2 outputs.

Figure 2 shows a detailed example of a single neuron within the hidden layer and the output layer. Each neuron u receives several inflows via the connections $c_1, \ldots, c_n$. Each of these inflows i is associated with a weight w_{iu}, which determines how strongly the values are influenced when they enter the neuron. The neuron u calculates the sum of all input values based on the following equation:

$$Q_u = \sum_{i=1}^{n} (w_{iu} c_u + w_{0u}) \tag{1}$$

In this equation, the term w_{0u} is also introduced, which is called bias. In addition, an activation function $q(x)$ is applied to the value Q_u to determine the final output value of the neuron based on its inputs[8].

[5] Forouzanfar et al. 2010, p. 2.
[6] Dawson and Wilby 1998, p. 49.
[7] Forouzanfar et al. 2010, p. 2.
[8] Dawson and Wilby 1998, p. 50.

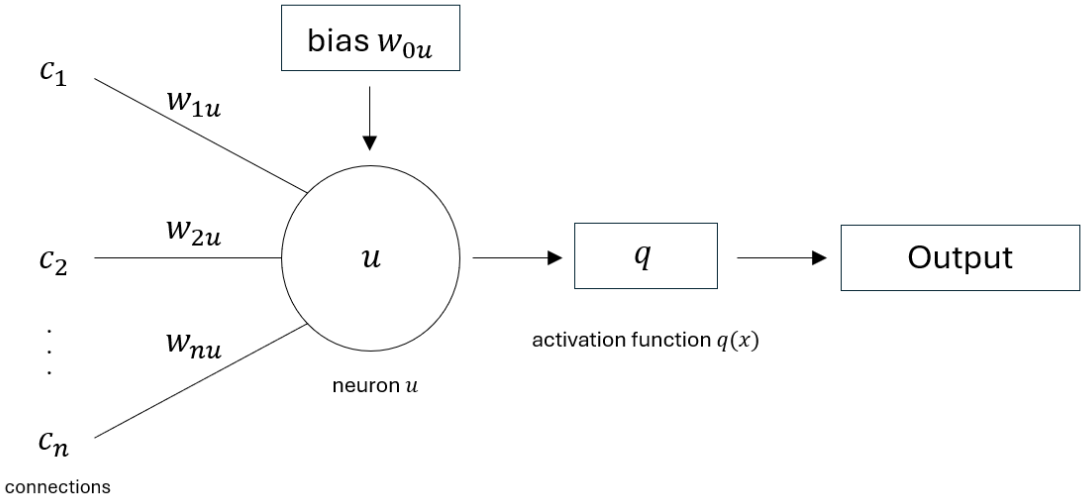

Figure 2: Detailed view of a neuron inside of an ANN.

This activation function can be linear, discrete or another distribution function ([15], p. 50). A sigmoid function is typically used in the hidden layers, while the output layer often uses the ReLU function. The sigmoid function in the hidden layers supports the capture of complex and non-linear relationships between inputs and outputs, whereas the linear structure of the output layer allows the outputs to cover an arbitrary range of values ([15], p. 49). The formulas for the sigmoid function[9] and the ReLU function[10] are shown in Formula 2 and Formula 3:

$$q_{\text{sig}}(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

$$q_{\text{relu}}(z) = \max(0, z) \tag{3}$$

The training process plays a crucial role in the overall performance of the neural network. It describes the process of determining the optimal weights and biases that connect the neurons by providing the network with a set of examples that demonstrate to the network what behavior is expected of it ([15], p. 50). For this purpose, a loss function is defined, which is usually represented by the mean square error (MSE) between the model prediction and the expected output. Minimization is achieved by calculating the gradient using the backpropagation technique ([14], p. 2). Backpropagation is an algorithm that uses error feedback to gradually adjust the weights in the network based on the gradient of the loss function. This method can be applied either in batch mode, where the weight updates occur after the entire training dataset has been processed, or in online mode, where the weights are adjusted after each individual training example ([14], p. 2). In order to use the backpropagation algorithm for the training process, the function used must have the property of being differentiable everywhere. Although the ReLU function is differentiable for $x \neq 0$, it is not differentiable at $x = 0$. In contrast, the sigmoid function fulfills this criterion and is therefore frequently used in most applications of neural

[9] Goodfellow 2016, p. 191.
[10] Goodfellow 2016, p. 189.

networks with a feedforward architecture ([15], p. 50).

In practice, a step is often introduced in which the weights and biases of a network are first initialized to small random values before training begins. A common rule of thumb is that these values should be chosen in the range $\left(-\frac{2}{n}, \frac{2}{n}\right)$, where n is the number of inputs to a neuron. If the initial random weights are not set within this range, network learning can be slowed down as extreme initial positions on the sigmoid function can limit the ability of the training algorithm to adapt the weights ([15], p. 50).

4 Extreme Gradient Boosting (XGBoost)

4.1 Fundamental Tree Algorithms

According to the evolution of tree algorithms, starting from basic techniques like decision trees, bagging, boosting, and gradient boosting, the XGBoost algorithm was developed and has been applied in many real-life sectors such as finance, healthcare, and retail. We will go through some basic definitions before diving deep into the concept of XGBoost.

4.1.1 Decision Trees

A **Decision Tree**[11] is a supervised learning algorithm used for classification and regression tasks in machine learning. It works by splitting the data into subsets based on feature values, forming a tree-like structure where each node represents a decision rule, and each leaf represents an outcome or prediction.

The terminology of a Decision Tree is as follows:

- The process starts with the **Root Node**, which represents the entire dataset and contains the first decision rule.
- The Root Node is split into one or more **Decision Nodes**, which represent intermediate decisions or conditions within the tree.
- **Decision Nodes** can be further split into other Decision Nodes or **Leaf Nodes**.
- When further splitting is not possible, the nodes become **Leaf Nodes**, representing the final outputs.
- The outcome of a decision rule is referred to as a **Branch** or **Sub-Tree**.

4.1.2 Ensemble Learning

Ensemble Learning[12] is an approach that trains different models (called learners) to solve the same problem. These learners do not need to be perfect, as they will be combined to build a stronger learner. This process continues iteratively, where each new learner improves upon the previous ones. This approach enhances performance and increases the accuracy of predictions. **Bagging** and **Boosting**[13] are two popular techniques related to ensemble learning. In bagging, each model is trained independently and in parallel using different bootstrap samples of the same dataset. Each model makes its own predictions, and the final decision is made by averaging the individual predictions (for regression) or by majority voting (for classification). On the other hand, in boosting, multiple models are trained sequentially. Each new model is designed to correct the errors made by the previous models. This means that the models essentially learn from their mistakes.

[11] Sebastiani, Abad, and Ramoni 2005, p. 165.
[12] Lu and Zhou 2022.
[13] Lu and Zhou 2022.

4.1.3 Gradient Descent

Gradient Descent[14] is an optimization algorithm used to minimize a model's loss function by iteratively adjusting the model's parameters (weights and biases). The goal is to find the parameter values that result in the lowest possible loss, thereby improving the model's predictions. Starting from a random position, the gradient descent method tries to find the steepest step toward the lowest point, where the loss function value is minimized. This approach is described by the equation:

$$p_{n+1} = p - \gamma \nabla f(p), \tag{4}$$

where p_{n+1} is the next position, p represents current posision, $\nabla f(p)$ is the direction of the steepest descent, which is the partial derivative of the function f at the current position p. The learning rate γ scales the gradient and controls the step size. A smaller learning rate results in slower convergence of gradient descent and may cause it to reach the maximum number of iterations before finding the optimal point.

4.2 Definition of XGBoost

Extreme Gradient Boosting (XGBoost) has become a popular machine learning algorithm due to its scalability, accuracy, and computational efficiency. It has evolved through many iterations of tree-based algorithms, including decision trees, random forests, and gradient boosting. XGBoost algorithm combines the principles of ensemble learning, boosting, and gradient descent, using decision trees as base learners. It is widely used for supervised learning tasks, such as regression and classification. The main functions of the XGBoost algorithm consist of handling complex relational data, using regularization techniques to prevent overfitting, and employing parallel processing for efficient computation. As an ensemble learning method, XGBoost systematically combines the predictive power of multiple learners, typically decision trees. The final result is a single model that aggregates the outputs of these individual learners.

4.3 Methodology of XGBoost

The methodology of XGBoost[15] begins with initial trees (weak learners) and sequentially adding many additional trees to form an ensemble. Each new tree is trained to correct the errors of the previous trees, with the goal of improving overall prediction performance. This boosting process iteratively minimizes the loss function by combining the outputs of all trees in the ensemble. The mathematical foundation of XGBoost is based on the principles of gradient boosting. The core idea is to improve the accuracy of the objective function. To understand the complexity behind the XGBoost algorithm, we should start with a simple formula for the target objective:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k). \tag{5}$$

Here l is a loss function that measures the difference between the prediction $\hat{y}_i$ and the target y_i. For example, in a regression task, a common loss function is the Mean Squared Error (MSE):

[14] Kwiatkowski 2021.
[15] Chen and Guestrin 2016.

$$l(\hat{y}_i, y_i) = (\hat{y}_i - y_i)^2.$$

The term Ω is regularization term that penalizes the complexity of the model. Intuitively, we can conclude from this formula that the target objective is built from the sum of the loss functions and the regularization terms. This means that the new loss function is updated and improved from the previous loss functions, adjusted by the regularization term. In each iteration t, a new tree is added to the model to minimize the objective function. The prediction at the t-th iteration is given by:

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i).$$

Over n training sample, the equation 5 can then be rewritten as:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t). \tag{6}$$

Where f_t represents a new tree that is added to the model to improve the current prediction $\hat{y}_i^{(t-1)}$ in each iteration t. Unlike traditional models, where parameters are typically continuous values, XGBoost builds the model in an additive manner because it is structured from many decision trees. This means that the optimization functions are neither differentiable nor continuous. Therefore, a second-order Taylor series expansion is used to address this problem. This approach makes the optimization manageable by turning a potentially complex and non-linear objective into a simpler form. XGBoost objective using second-order Taylor approximation is as follows:

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t),$$

where

$$g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \quad \text{and} \quad h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

are first and second order gradient statistics on the loss function, respectively. We derive a simpler version of the objective by removing the constant terms at step t:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t). \tag{7}$$

We define $I_j = \{i | q(x_i) = j\}$ as the set of instances or observations corresponding to leaf j, where $q(x)$ represents a function that assigns an input x_i to a specific leaf j. This implies that all instances x_i assigned to the same leaf j are group together. For example, if instances x_1, x_4, x_5 belong to leaf 2, then $I_2 = \{1, 4, 5\}$. We expand regularization term Ω as:

$$\Omega = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2,$$

where γT represents the penalty for the number of leaves (T) in the model, and $\frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$ accounts for the regularization on leaf weights w_j. The equation can be transformed as follows:

$$\begin{aligned}\tilde{\mathcal{L}}^{(t)} &= \sum_{i=1}^{n}\left[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)\right] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T}\left[\left(\sum_{i\in I_j} g_i\right) w_j + \frac{1}{2}\left(\sum_{i\in I_j} h_i + \lambda\right) w_j^2\right] + \gamma T.\end{aligned}$$

Based on the results from Tianqi Chen[16], the scoring function used to measure the quality of a tree structure q is given by:

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{\left(\sum_{i\in I_j} g_i\right)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T. \tag{8}$$

The goal of the scoring function is to find a tree structure that minimizes the overall loss while preventing overfitting. This is achieved by balancing the fit to the training data with the complexity of the tree.

Although XGBoost uses decision trees as base learners, it is still considered a "black box". This is especially true when the model involves a large number of features and uses many decision trees as learners. The gradient boosting process can be time-consuming and complex, making it challenging for users to interpret and understand the model's behavior.

[16] Chen and Guestrin 2016.

5 Post Hoc Explainability Methods

Post-hoc methods are approaches that solve the black box problem by making models interpretable without revealing their internal structures. They can be used flexibly, enable different representations to explain a model and can provide several perspectives on the same model. There is always a trade-off between the accuracy and comprehensibility of the explanations. Post-hoc methods can provide deeper insights into the data distribution that other approaches miss out on, and their combined use enables comprehensible explanations even for complex models. There are global post-hoc methods that describe the entire model behaviour and local post-hoc methods that focus on individual predictions. SHAP (Shapley Additive Explanations) combines both approaches, which will be explained in detail later on.

The correct interpretation of the output of a predictive model is crucial. In practice, simpler models such as linear models are often used in this context as they are easier to interpret, although they tend to be less accurate than more complex models. However, with the increasing availability of big data from diverse systems, the use of complex models is increasing, which increases the need for improved interpretation methods. Models are traditionally considered interpretable if their overall behavior can be clearly summarized. In contrast, complex models do not allow a concise summary, but the relationships can be more easily understood when looking at individual predictions. Newer model-agnostic methods exploit this property by representing the behavior of complex models specifically for individual predictions[17].

5.1 SHAP

The features of a data set contribute to the predictive performance of a model to varying degrees. There are various methods for determining feature relevance that quantify the contribution of individual features to model prediction by considering additional factors such as type, robustness, comprehensibility, and the quality of the explanations. The explanations can shed light on the global behavior of a model as well as locally on the prediction of individual data instances. Each method has specific advantages and disadvantages, and some are model-specific or optimized for specific model methods. A number of these methods are presented below[18].

A precise prediction and the corresponding correct interpretation creates trust, provides insights for model improvement and promotes understanding of the process[19]. An agnostic approach to machine learning means that a model f makes no specific assumptions about the underlying data distribution or the relationship between input and output data. Instead, it is assumed that the data can be arbitrarily distributed, and the goal is to develop an explanatory model g that performs as well as possible in generalization under these uncertain conditions[20]. The input data x from the original model is often difficult to interpret and the underlying model, such as a neural network, is difficult to explain[21]. Therefore, the input data x is transformed into a new, simplified set x' by a simple mapping $x = h(x')$. For example, input data consisting of a vector of words can be simplified as present or not present (binary).

[17] S. Lundberg and Lee 2016, p. 1.
[18] Kamath and Liu 2021, p. 184.
[19] S. Lundberg 2017, p. 1.
[20] Gianfagna and Di Cecco 2021, p. 116.
[21] S. Lundberg 2017, p. 2.

For feature attribution, taking these steps into account, the explanatory model $g(x')$ of additive methods can be represented as

$$f(x) = g(x') = \Phi_0 + \sum_{i=1}^{n} \Phi_i x'_i \tag{9}$$

where n is the number of features and Φ_0, a constant resulting from the feature-independent model output, i.e. $\Phi_0 = f_x(\emptyset)$. Methods that use explanatory models according to equation 9 assign a value Φ_i to each feature; the sum of these values then results in the output of the model f. Some such methods are presented below[22].

A solution for the equation 9 is provided by the Shapley values with their properties of local accuracy, zero property and consistency. The local accuracy ensures the consistency of the outputs between the original and the explanatory model. Local accuracy exists when $x = h(x`)$, where $h$ is a mapping function and $x`$ is the simplified transformation of the original input data x[23]. The zero property means that irrelevant features have no influence on the model prediction, i.e. Φ_0, if $x'_i = 0$[24]. They are also consistent in the sense that a feature that makes a larger contribution to the model is assigned a higher Shapley value. They have great potential as they are the only method that is a solution for equation 9 and at the same time have the three characteristics and thus form the basis for many other explanatory models[25].

Shapley values are based on game theory and represent a valuable method for evaluating feature relevance. They are based on the principle of distributing the importance of the characteristics among the characteristics involved by applying cooperative game theory. Shapley values are a concept named after Lord Shapley, who introduced this concept in 1951 to find solutions in cooperative games. Each characteristic is a player in a game whose goal is to achieve a collective gain as a group and distribute the gain fairly among the players based on their contribution to the overall gain[26]. Shapley values can be described by taking the average over the set of all possible coalitions of traits to find the marginal contribution of each trait[27]. Lord Shapley showed in 1953 that the Shapley value is efficient, symmetric, neutral and additive, which implies a fair distribution[28].

To make the analogy with machine learning and Explainable AI (XAI), *profit* is replaced by *prediction* at this point. The game-theoretical idea described above can be transferred to machine learning as follows: The aim of an ML model is to determine the relevance of the features based on their contribution to the prediction. Shapley values therefore show which features have contributed more to a particular prediction.

[22] S. Lundberg 2017, p. 2.
[23] Mangalathu, Hwang, and Jeon 2020, p. 2.
[24] S. Lundberg 2017, p. 3.
[25] S. Lundberg and Lee 2016, p. 3.
[26] Kamath and Liu 2021, p. 191.
[27] Gianfagna and Di Cecco 2021, p. 94.
[28] Kamath and Liu 2021, p. 191.

For a model f with n features, the Shapley value Φ_j for the feature x_j to be explained and the coalition K, in which x_j is not included, can be generalized by

$$\Phi_j(f) = \sum_K \frac{|K|!(n - |K| - 1)!}{n!} \Big[f(K \cup \{x_j\}) - f(K) \Big]. \tag{10}$$

The posterior term describes the difference between two models, one of which was trained with feature x_j and the other without feature x_j, i.e. only with K[29]. Since the influence of omitting a feature depends on the other features in the model, the posterior term is run with all possible coalitions and the Shapley value is the average of the marginal contributions of all possible coalitions[30]. This formula shows that the Shapley value answers the question of how the marginal contribution changes when x_j is added to all coalitions K. The prediction of the additive model can therefore be expressed as the sum of all Shapley values and the expected model prediction, i.e.[31]

$$f(x) = \sum_{m=1}^{n} \Big[\Phi_j(f) + E[f(x)] \Big]. \tag{11}$$

The SHAP (*SHapley Additive exPlanations*) is a standardized measure of feature relevance. The value of the SHAP corresponds to the Shapley value of a conditional expected value of f, taking into account the possible coalitions, i.e.

$$f(X) = E[f(X) \mid K]. \tag{12}$$

Thus, they assign a value to each feature that describes the change in the expected model prediction when that feature is included. The SHAP value describes how to get from the base value $\mathbb{E}[f(0)]$, which would be assumed if no features were known, to the output of the model $f(X)$[32].

The number of possible coalitions K mentioned above is represented by 2^{n-1}, where n is the number of features. From this it can be seen that this value increases exponentially and with an increasing number of features this implies that the computational effort of the Shapley values, and thus also that of the SHAP, also increases exponentially. Depending on the type of model (ANN, decision trees, etc.), there are various methods for approximating the SHAP. Two methods, the KernelSHAP and the DeepSHAP, are explained in more detail below[33].

5.1.1 KernelSHAP

In order to understand the characteristics of the KernelSHAP, the term "LIME" must first be introduced, as the KernelSHAP is a special form of LIME[34].

[29] Kamath and Liu 2021, p. 191.
[30] S. Lundberg 2017, p. 3.
[31] Kamath and Liu 2021, p. 191.
[32] S. Lundberg 2017, p. 5.
[33] Gianfagna and Di Cecco 2021, p. 99.
[34] Gianfagna and Di Cecco 2021, p. 101.

A local explanatory model refers to explanations that focus on individual predictions or specific data instances and helps to understand why a model made a particular decision for a particular data point, whereas a global explanatory model is concerned with how the model works in general and what patterns it has learned[35]. The former is known to perform better than the latter[36].

Local replacement models attempt to explain a prediction of an explanatory model g for individual data instances x without understanding the entire model. In doing so, they exhibit an important property called *local fidelity*. Local fidelity refers to capturing the behavior of the model in the environment of the data instance to be explained. One such model is the so-called LIME (*Local Interpretable Model-Agnostic Explanations*). It is a model-agnostic method for explaining individual predictions in the neighborhood of the prediction for models of any kind[37]. To learn a local explanation, the LIME method approximates the decision boundary of the model around a specific data instance using a linear model such as formula 9[38] and is thus an additive method of feature attribution[39].

To find such a linear model, the following optimization problem is solved for the model f, the explanatory model g, which comes from a set of interpretable models (e.g. linear, logistic models, decision trees), and the data instance x to be predicted:

$$\mathcal{L}(g) = L(f, g, \pi_x(z)) + \Omega(g). \tag{13}$$

The function $\pi_x(z)$ is an exponential function that weights a sample z and decreases as the distance to data instance x increases. The sample z describes a new class of data instances that are generated by random disturbances such as Gaussian noise[40] and can therefore be interpreted[41]. In this way, the model can be little influenced by data points that originate from the same source (sampling noise)[42]. The loss function $L(f, g, \pi_x(z))$ indicates how well or poorly the model f was approximated by the explanatory model g in the vicinity of x and the complexity term $\Omega(g)$ limits the number of parameters of the model g in order to keep the complexity of the model low. The more complex the explanatory model g, the greater the value of Ω. The interpretability of predictions can be limited by limiting the complexity term without violating local fidelity[43].

As with the additive feature attribution methods, the LIME method uses a mapping function to represent the data instances by using a binary vector to represent the presence or absence of a feature. However, these interpretable simplifications of the data instances result in important information of the data being lost in the mapping

[35] Kamath and Liu 2021, p. 12.
[36] Gianfagna and Di Cecco 2021, p. 99.
[37] Kamath and Liu 2021, p. 201.
[38] Mishra, Sturm, and Dixon 2017, p. 538.
[39] S. Lundberg 2017, p. 2.
[40] Kamath and Liu 2021, p. 201.
[41] S. Lundberg 2017, p. 2.
[42] Kamath and Liu 2021, p. 201.
[43] Gianfagna and Di Cecco 2021, p. 101.

process or limiting the interpretability of the model[44].

The LIME method is characterized by its high flexibility and the ability to generate easily understandable models. Nevertheless, it shows weaknesses in the explanation of models with nonlinear behavior in the local environment of the data instance to be predicted. The samples z generated by adding random perturbations are merely representations of the actual data instances. These samples do not fully reflect the data instances x and can lead to different explanations for data instances that are in close proximity to each other[45].

As already explained, the determination of Shapley values is challenging, as they analyze the marginal contribution of each individual feature to all possible coalitions. As the number of features increases, the computational effort increases exponentially. Accordingly, an efficient approximation of the Shapley values is all the more important[46]. The KernelSHAP provides such an approximation and is the only local (linear) replacement model specifically designed for Shapley values[47]. Like the LIME method, it uses a weighting factor π_x and approximates Shapley values in the local neighborhood of the data instance to be predicted, sampling over all possible coalitions K by applying a linear additive model to the original model f[48].

The calculation of KernelSHAP corresponds to the fitting of a linear model including the Shapley values by minimizing the objective function

$$\Psi = \min_{\Phi_i,\ldots,\Phi_M} \left[\sum_K \left(f(K) - \sum_{x_j \in K} \Phi_j \right)^2 \pi(K) \right] \tag{14}$$

with the weighting factor

$$\pi(K) = \frac{(n-1)}{\binom{n}{|K|} |K|(n-|K|)}. \tag{15}$$

In contrast to the LIME method, π here depends on the coalitions K and not on the simplified data instances. Formula (15) shows that π is a weighting function that assigns a higher weight to the smallest and largest coalitions, as these provide the most information about the feature relevance. A coalition consisting of a single feature is just as informative as a coalition comprising all but one feature[49]. Due to the linear form of f and the fact that Ψ is a quadratic loss function, linear regression can be applied to solve 14 and in particular to determine $\Phi_j(f)$[50].

[44] Kamath and Liu 2021, p. 201.
[45] Kamath and Liu 2021, p. 202.
[46] Covert and Lee 2020, p. 3.
[47] Gianfagna and Di Cecco 2021, p. 101.
[48] Kamath and Liu 2021, p. 193.
[49] Kamath and Liu 2021, p. 193.
[50] S. Lundberg 2017, p. 6.

5.1.2 DeepSHAP

In the field of neural networks, understanding and addressing gradient-related issues is crucial for improving model performance and interpretability. The shattered gradient problem highlights how the correlation between gradients in standard neural networks diminishes exponentially with depth. This issue is intensified by the use of activation functions like ReLU, which can lead to locally flat regions and discontinuous gradients. To mitigate these challenges, methods such as Deep Learning Important Features (DeepLIFT) have been developed. DeepLIFT offers a way to ensure that the gradient flow through activation functions remains undistorted, providing a partial solution to the shattered gradient problem. This method, along with DeepSHAP, which integrates Shapley values for enhanced feature attribution, is discussed below to offer insights into improving the interpretability and reliability of deep learning models.

DeepLIFT (Deep Learning Important Features) is a recursive, local explanation method that decomposes a neural network prediction for each individual data instance by backpropagating the contributions of the nodes through the neural network[51]. This method compares the activation of each node with its reference activation and evaluates the relevance of each contribution[52]. For a given prediction, this method provides local explanations by quantifying the contributions. This is done by calculating the difference in output compared to a reference output, taking into account the difference in input compared to a reference input[53].

Assume a neural network with the nodes of the input layer $\{x_1, x_2, \ldots, x_n\}$, and a node of the output layer t is given. If t‘ is the reference activation of t, then the difference Δt can be defined as the *difference-from-reference* by the value $\Delta t = t - t'$. Furthermore, let the output of the model of a certain node $f(x)$ and the corresponding reference value $f(x')$, be given, assuming that the reference of the node is the activation function of the reference input. Here, t corresponds to the value of f; analogously for their reference values[54].

The DeepLIFT method assigns contribution values to the Δx_i values as follows:

$$\Delta t = f(x) - f(x') = \sum_{i=1}^{n} C_{\Delta x_i \Delta t} \tag{16}$$

where $\Delta x = x - x'$ is the *difference-from-reference* and the contribution value $C_{\Delta x_i \Delta t}$ is the contribution of each individual node x_i. Formula (16) therefore shows that the contribution value $C_{\Delta x_i \Delta t}$ answers the question of how the change in Δx influences the change in Δt[55].

The multiplier 17 for an input node x with difference-from-reference Δx and an output node t with difference-from-reference Δt is defined as the quotient between the contribution Δx to Δt and Δx, which follows a chain rule (Formula 2.10) that

[51] Kamath and Liu 2021, p. 237.
[52] Gianfagna and Di Cecco 2021, p. 131.
[53] Shrikumar, Greenside, and Kundaje 2017, p. 3.
[54] Shrikumar, Greenside, and Kundaje 2017, p. 3.
[55] Kamath and Liu 2021, p. 238.

can be used to propagate the contribution value $C_{\Delta x_i \Delta t}$ back layer by layer in the neural network[56].

$$m_{\Delta x \Delta t} = \frac{C_{\Delta x \Delta t}}{\Delta x} \tag{17}$$

$$m_{\Delta x_i \Delta t} = \sum_j m_{\Delta x_i \Delta y_j} \, m_{\Delta y_j \Delta t} \tag{18}$$

Recall equation 9 of the additive method for feature attribution. If Φ_i is replaced here by the contribution value $C_{\Delta x_i \Delta t}$ and Φ_0 by $f(x')$, the values still fulfill equation 9 and the DeepLIFT method would thus be another model of the additive methods for feature attribution[57]. DeepLIFT uses a linear composition rule, which corresponds to a linearization of the nonlinear components of a neural network[58]. DeepLIFT has a methodological link to Shapley values. Shapley values quantify the marginal contribution of each feature by averaging over all possible coalitions. If the exclusion of a feature corresponds to the reference value, DeepLIFT can be regarded as an efficient method for approximating the Shapley values[59].

DeepSHAP can be seen as an extension of the KernelSHAP method by utilizing the internal structure (architecture) of a deep neural network and can approximate Shapley values by applying the chain rule of the DeepLIFT method. In analogy to the multiplier and the chain rule of the DeepLIFT method, a corresponding approach can also be derived for the DeepSHAP method, taking into account the SHAP values, and equations for the multiplier and the chain rule are formulated[60]:

$$m_{x_j, f_j} = \frac{S_i(f_j, x)}{x_j - \mathbb{E}(x_j)} \tag{19}$$

$$m_{x_j, f_j} = \sum_j m_{x_j, y_j} \, m_{y_j, f_j} \tag{20}$$

The DeepSHAP method determines SHAP values for small parts of a large network and transfers them to the entire network by recursively backpropagating the DeepLIFT multipliers using the chain rule 20[61] and assuming that the network is linear[62]. Unlike the DeepLIFT method, DeepSHAP approximates the reference value x' by taking the average over data instances of the background dataset. This allows SHAP values to be estimated that add up to the difference between the expected model output formed on the background data and the actual model output $f(x) - \mathbb{E}[f(x)]$[63]. The linear composition rule mentioned above enables a fast ap-

[56] Shrikumar, Greenside, and Kundaje 2017, p. 3.
[57] S. Lundberg 2017, p. 3.
[58] S. Lundberg 2017, p. 7.
[59] Kamath and Liu 2021, p. 238.
[60] Kamath and Liu 2021, p. 240.
[61] Kamath and Liu 2021, p. 238.
[62] S. Lundberg 2017, p. 7.
[63] Kamath and Liu 2021, p. 238.

proximation of the values for the entire model, since SHAP values for the small parts of the network can be determined efficiently using the chain rule if they are linear[64].

5.1.3 TreeSHAP

TreeSHAP is a specialized implementation of SHAP designed specifically for decision tree-based models like the XGBoost. It leverages the structure of decision trees to compute SHAP values efficiently, significantly reducing computational complexity compared to general SHAP algorithms. TreeSHAP systematically calculates the marginal contributions of individual features in a tree model[65].

Direct computation of these values is computationally expensive, as it involves evaluating all possible subsets K. TreeSHAP addresses this challenge by exploiting the hierarchical structure of decision trees. Decision trees split the feature space through a sequence of hierarchical decisions. Each node in a tree applies a condition (e.g. $x_j < t$) that divides the data into two subsets. TreeSHAP utilizes this structure by computing how much of a prediction in a leaf can be attributed to each feature. Instead of explicitly calculating the contribution for every subset K, it propagates contributions efficiently through the tree. The central idea is to distribute the prediction responsibility of each feature across the tree structure, leveraging the relationships between parent and child nodes. Each path through the tree represents a sequence of conditions that determine whether a data point reaches a particular leaf[66].

For any leaf l in the tree, the probability that a sample x reaches the leaf P_l is calculated as the product of the probabilities of all conditions along the path to l:

$$P_l = \prod_{k \in \text{Path}(l)} P_k \tag{21}$$

where P_k is the probability of the split condition at node k being satisfied for the data point. For a node v, let the contribution to the prediction be split into portions that propagate down to v_L and v_R. The recursive formula for updating contributions is:

$$\phi_j(v) = P(v_L) \cdot \phi_j(v_L) + P(v_R) \cdot \phi_j(v_R), \tag{22}$$

where $P(v_L)$ and $P(v_R)$ are the probabilities of the data reaching the left or right child, respectively. If the node v itself uses the feature x_j in its splitting condition, the recursive formula adjusts for this by calculation the expected contribution of x_j to the split. The total contribution of x_j is updated based on the marginal effect of splitting at v, given the feature's presence in the decision. The recursive process is applied to all nodes in all trees of the model. Once the contributions from all paths

[64] S. Lundberg 2017, p. 7.
[65] Gianfagna and Di Cecco 2021, p. 107, 108.
[66] S. M. Lundberg et al. 2019, p. 4 - 6.

have been computed, the SHAP value ϕ_j for each feature x_j is obtained by summing the contributions across all relevant paths:

$$\phi_j = \sum_{\text{all paths}} \phi_j(\text{path}) \tag{23}$$

This approach shows that TreeSHAP can efficiently assign responsibility for the prediction to individual features, leveraging the structure of decision trees to avoid redundant computations[67].

5.2 Accumulated Local Effects (ALE) Plot

In many applications of supervised learning, it is crucial to understand and visualize how predictor variables influence the predicted response. However, black box supervised learning models (such as complex trees, neural networks, boosted trees, random forests) often fall short in this aspect due to their lack of interpretability or transparency. There are several methods for visualizing predictor effects in black box models, namely the **Partial Dependence Plot (PDP)**, the **Marginal Plot (M-Plot)** and the **Accumulated Local Effects (ALE) Plot**. In our study, we will use ALE Plot to explain the effect of features on the model's predictions. However, it will be simpler if we first review how PDP and M-Plot work, identify their limitations, and explain how the ALE method addresses these issues.

5.2.1 PDP and M-Plot

Partial Dependence Plot (PDP)[68] is one of the most popular approaches for interpreting black-box supervised learning models. The idea behind PDP is straightforward: take one or more features of interest, vary their values, and keep all other features constant to observe how the model's predictions change. By repeating this process multiple times and averaging the predictions, we obtain the PDP.

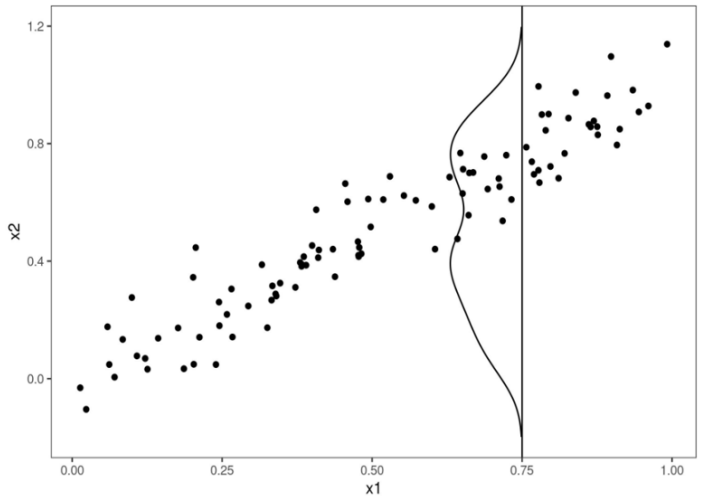

Figure 3: PDP showing the effect of x_1 on predictions.[69]

[67] Yang 2021, p. 3, 4.
[68] Molnar 2020.

As shown in Figure 3, PD Plot focuses on the global effect of a feature on the prediction, averaging out the effects of all other features. However, this approach operates under the **assumption that all features are independent**. In practice, this assumption often does not hold, as real-world data frequently involve dependent features. This can lead to misleading interpretations, especially in models with complex interactions between features. For example, in a house pricing scenario where x_1 and x_2 represent house size and number of rooms, respectively, fixing the value of x_1 (house size) to 50 m^2 might result an unrealistic scenario where the model considers a house with $x_2 = 20$ rooms within just 50 m^2. Such situations arise because the PDP assumes feature independence, ignoring the natural relationships or constraints between features.

To address the problem of PD Plots, **Marginal Plot** (**M-Plot**)[70] focuses on the local effect of a feature on the prediction by using conditional distribution in situations where features are correlated, as shown in Figure 4. The idea is to calculate the average predictions while conditioning on the value of the feature of interest.

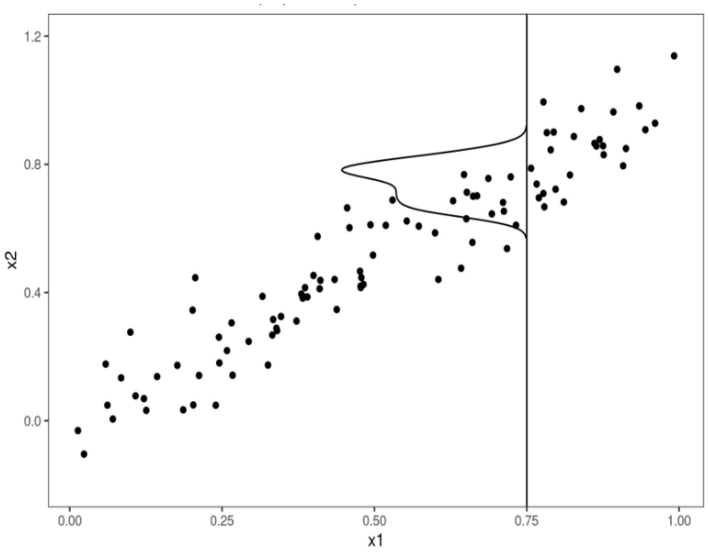

Figure 4: M-Plot showing the effect of x_1 on predictions.[71]

However, this approach also has limitations. Returning to the house pricing example above, if we calculate the average predictions for all houses around 50 m^2, we estimate the combined effect of the house size and the number of rooms due to the correlation. Even if house size has no effect on the price of a house while the number of rooms does, the M-Plot will still show that increasing house size raises the house price. This occurs because averaging local predictions mixes the effects of the feature of interest with those of correlated features. In conclusion, M-Plots can help advoid unrealistic data instances, but they still mix the effect of a feature with the effects of all correlated features.

5.2.2 Introduction to ALE Plot

To address the limitations of PDPs and M-Plots, **Accumulated Local Effects (ALE) plots**[72] provide a robust alternative for isolating the effect of a feature while mitigating the influence of correlated variables. Unlike PDPs, which assume feature

[70] Molnar 2020.
[72] Apley and Zhu 2020.

independence, ALE plots compute local effects by examining how predictions change when the feature of interest varies within small, localized intervals of its observed range. This approach conditions on realistic data distributions, ensuring that the contributions of correlated features are effectively separated. By accumulating these localized effects across the feature's range, ALE plots provide a comprehensive and unbiased visualization of the feature's true impact on the model's predictions. This makes them a powerful and reliable tool for interpreting machine learning models, particularly in the presence of correlated features.

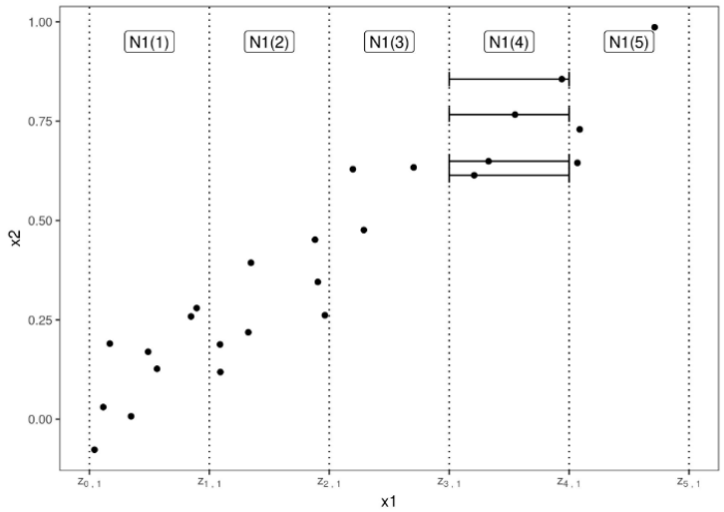

Figure 5: Calculation of ALE for feature x_1, correlated with x_2.[73]

5.2.3 Methodology of ALE

As mentioned, the ALE plots calculate the differences in predictions rather than averaging them. To achieve this, the feature of interest is divided into intervals. There are two ways to split data points into intervals: *equal-width intervals* (where intervals have the same width) or *equal-frequency intervals* (where each interval contains the same number of data points). A technique using quantiles of the distribution is applied to make sure that the number of data points within each interval is the same, However, this approach may result in unusual ALE plots, making them difficult to analyze due to abrupt changes in interval boundaries[74].

For the mathematical formula of ALE plot[75] , consider a model f with two features x_1 and x_2. The ALE function for feature x_1 is described as follows:

$$\text{ALE}(x_1) = \int_{\min(x_1)}^{x_1} \mathbb{E}[f(x_1, x_2) \mid x_1 = z]\, dz - C, \tag{24}$$

where the constant C is chosen to center the plot such that the mean effect is zero:

$$C = \mathbb{E}[\text{ALE}(x_1)].$$

The formula for ALE does not primarily involve dividing the dataset into intervals. It calculates the local effects and then accumulates the prediction results from these local effects, starting from the smallest observed value of x_1 to the current

[74] Molnar 2020.
[75] Apley and Zhu 2020, p. 4.

value of x_1. At the final step, the accumulated predictions are centered by subtracting the mean effect of ALE. This centering process helps interpret the coefficients in a regression model more easily, as the data becomes balanced around a mean of zero. The expression $x_1 = z$ represents the value of the feature x at a specific point z within an interval. The expectation $\mathbb{E}[f(x_1, x_2) \mid x_1 = z]$ can be expanded using the definition of conditional expectation:

$$\mathbb{E}[f(x_1, x_2) \mid x_1 = z] = \int p(x_2 \mid z) f(x, z) dx_2,$$

where $p(x_2 \mid z)$ is the conditional probability density of x_2 given by $x_1 = z$. Here, we vary x_1 while keeping other features (in this case, x_2) at realistic values that are observed in the dataset. This help us understand how the model's output f changes in response to changes in x_1. To capture this effect, partial derivative of f with respect to z is needed:

$$\int p(x_2 \mid z) f(x, z) dx_2 \to \int p(x_2 \mid z) \frac{\partial f(z, x_2)}{\partial z} dx_2.$$

Applying this into equation 24, we get:

$$\text{ALE}(x_1) = \int_{\min(x_1)}^{x_1} \int p(x_2 \mid z) \frac{\partial f(z, x_2)}{\partial z} \, dx_2 \, dz - C. \tag{25}$$

The **Accumulated Local Effects** (ALE) method shows how input features influence the prediction results of a neural network model. An ALE value greater than zero means that the feature increases the likelihood of the predicted outcome. Conversely, a negative ALE value indicates that the feature decreases the likelihood of the predicted outcome. If the ALE value is equal to zero, the feature has no effect on the prediction.

6 Results

In our analysis, we used ANN (Artificial Neural Networks) and XGBoost (eXtreme Gradient Boosting) to predict creditworthiness based on the *german credit data*. These models leverage advanced machine learning techniques to capture complex patterns in the data, offering high predictive accuracy. To ensure the interpretability of these predictions, we employed two prominent methods, ALE (Accumulated Local Effects) and SHAP (SHapley Additive exPlanations). ALE provides global explanations by illustrating the average effect of individual features across the dataset, offering insights into how specific variables, such as debt-to-income ratio or repayment history, influence the overall predictions. SHAP, on the other hand, enables both global and local explanations. It quantifies the contribution of each feature to individual predictions, allowing us to understand why a specific borrower received a high-risk or low-risk score. For instance, SHAP values might reveal that a borrower's recent credit utilization significantly contributed to their risk assessment, while their long-term repayment history had a mitigating effect.

By combining these methods, we achieve a comprehensive understanding of the predictive models. ALE highlights general trends and relationships in the data, while SHAP provides granular insights into individual cases. This dual approach ensures that we address both overarching and case-specific concerns, enhancing the interpretability of the predictions. The results of these analyses will be presented in the next chapter, where we delve deeper into the implications of our findings and their relevance to decision-making in creditworthiness assessment.

6.1 Modeling

This chapter focuses on the modeling of ANN and XGBoost, detailing the steps involved in building these models and optimizing their parameters for the given dataset. A crucial aspect of any ML pipeline is the dataset itself, as the quality and structure of the data directly impact the performance of the models. Therefore, an in-depth discussion of the dataset and the preprocessing methods applied, including data cleaning, feature selection, and normalization, is provided. Furthermore, the chapter evaluates the performance of both ANN and XGBoost, comparing their effectiveness based on key performance metrics. By analyzing accuracy, F1-score, and other relevant evaluation criteria, the study aims to determine which model is better suited for the given task. Effective modeling is essential in machine learning as it defines the ability of the model to generalize well to unseen data. A well-structured model ensures not only high predictive performance but also reliability and interpretability, which are crucial in real-world applications. Through this analysis, the study aims to provide insights into the strengths and limitations of ANN and XGBoost in predictive modeling.

6.1.1 Dataset and Preprocessing

The dataset used in this study is taken from Kaggle and describes the creditworthiness of clients in Germany. It consists of 1,000 observations across 21 variables. Most of these input features are categorical, representing different states or levels of a variable. The dataset includes basic information about borrowers, such as the credit amount, marital status, age, employment status, purpose of the loan, and whether the borrower works abroad. Some key variables that play a crucial role

in prediction include *Account_Balance*, which takes values of 1, 2, 3, or 4. A value of 1 indicates no account, 2 represents an account with no balance, 3 signifies an account with some balance, and 4 denotes an account with a substantial balance. Another important variable is *Guarantors*, which takes values of 1, 2, or 3, indicating whether an individual has no guarantors, individual guarantors, or organizational guarantors. Further information can be found at this link. The classification **target variable** *Creditability* represents whether an individual is able to obtain credit, as illustrated in Figure 6. A value of 1 indicates non-default (i.e., the individual is eligible for credit), while a value of 0 indicates default (i.e., the individual is not eligible for credit).

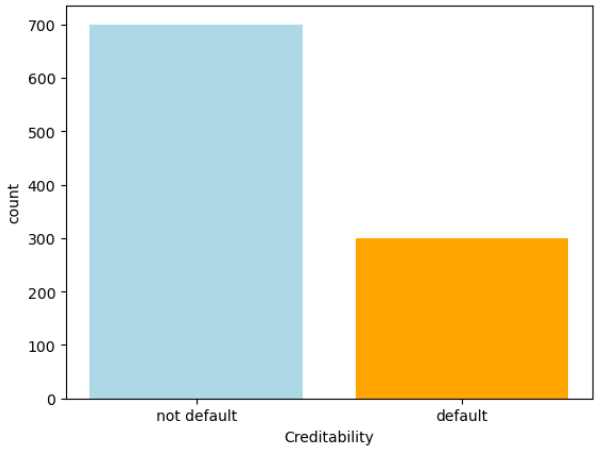

Figure 6: Number of Target Variable Observations.

Data imbalance typically occurs in classification tasks, and this dataset is no exception. As shown in Figure 6, the number of non-default observations is significantly higher than the number of default observations (700 vs. 300). To address this issue, a data preprocessing step is required. As we know from neural networks, features are initially assigned weights when they enter the model. The **Class Weights** method[76] is applied to adjust the weights of the classes during the loss function calculation. The majority class is assigned lower weights, while the minority class is assigned higher weights. This ensures that the model pays more attention to the minority class, helping to correct the imbalance. The class weights method is defined as follows:

$$w_c = \frac{N}{C \cdot n_c}, \tag{26}$$

where N represents total number of samples, C is number of classes and n_c represents number of samples in class c. The new class weights are then used to calculate the **Average Loss** of the loss function:

$$\text{Average Loss} = \frac{1}{N} \sum_{i=1}^{N} \left[- \sum_{c=1}^{C} w_c \cdot y_{i,c} \cdot log(p_{y_{i,c}}) \right], \tag{27}$$

where $y_{i,c}$ is label for class c for sample i and $p_{i,c}$ is the predicted probability for class c for sample i. The average loss value is optimized during the backward propagation phase of neural network training, updating the weights of the input features. In our study, the target variable has two classes (0 and 1), observed from 1000 clients. The

[76] Abhinav 2023.

number of samples per class is 300 for class 0 and 700 for class 1. The class weights are calculated as follows:

$$w_o = \frac{1000}{2 \cdot 300} = \frac{1000}{600} = 1.667,$$

$$w_1 = \frac{1000}{2 \cdot 700} = \frac{1000}{1400} = 0.714.$$

The average loss function used in backward propagation is:

$$\text{Average Loss} = \frac{1}{N} \sum_{i=1}^{N} \left[w_1 \cdot y_i \cdot \log(p_i) + w_0 \cdot (1 - y_i) \cdot \log(1 - y_i) \right].$$

Substituting our values:

$$\begin{aligned} \text{Average Loss} &= \frac{1}{1000} \left[700 \cdot (-1.667) \cdot \log(1 - p_i) + 300 \cdot (-0.714 \cdot \log(p_i)) \right] \\ &= \frac{1}{1000} \left[-1166.9 \cdot \log(1 - p_i) - 214.2 \cdot \log(p_i) \right]. \end{aligned}$$

The process then continues with forward propagation, repeating iteratively. In our dataset, there are many features of different types and the scale differences between them are too large (e.g., categorical features like *Account_Balance* with values 1, 2, 3 and 4 versus continuous features like *Credit_Amount*). These disparities can lead to feature imbalance, causing certain features to dominate the learning process. To mitigate this, the **StandardScaler()** method[77] is applied during data preprocessing. StandardScale is a preprocessing tool that normalizes input features by scaling them to have a mean of 0 and a standard deviation of 1. This standardization ensures that features are on a similar scale, improving model performance and preventing bias. By scaling the data, no single feature dominates due to its magnitude, ensuring fair contribution from all features.

6.1.2 ANN Model

The construction of the artificial neural network (ANN) involves 20 input features, which serve as the data fed into the model. The network contains three hidden layers, each consisting of 40 neurons, allowing the model to learn complex patterns and relationships within the data. The output layer has a single neuron, which represents the target variable, 'creditability.' The training loop runs for 100 epochs, meaning that the model will process the entire dataset 100 times, gradually adjusting its weights and biases to minimize the error and improve its predictive performance.

Fig. 7 illustrates the loss (error) over the number of epochs during the training of a neural network. On the x-axis, we see the number of epochs, representing the iterations during training, while the y-axis shows the loss value, which measures the model's performance. At the start of the training process (Epoch 0), the loss is relatively high (approximately 0.36), indicating poor initial predictions by the model. As the training progresses, the loss steadily decreases, reflecting the model's improvement in making predictions. By the end of the training (around 100 epochs),

[77] Thakker and Buch n.d.

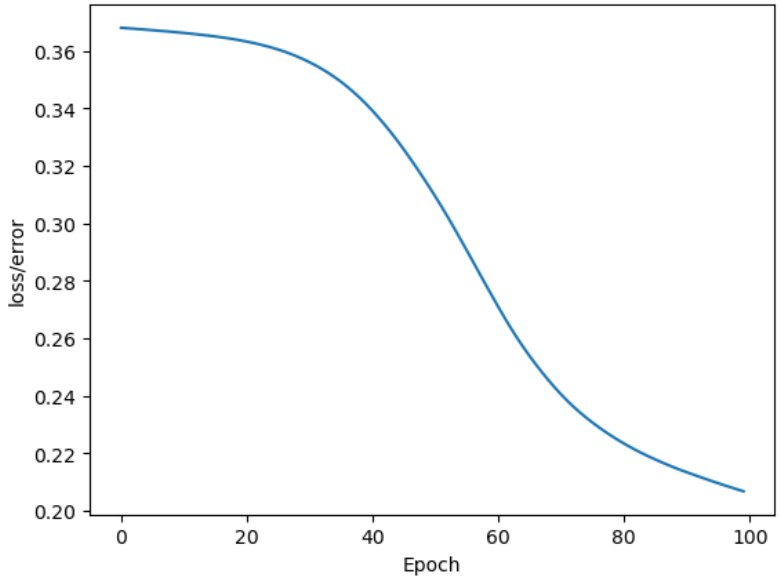

Figure 7: Illustration of the loss (error) over the number of epochs during the neural network training.

the loss converges to a stable, low value of approximately 0.20, suggesting that the model has effectively learned from the data. This smooth, downward trajectory of the loss is typical of a well-functioning training process. There are no signs of overfitting, which would manifest as an increase in loss on validation data after a certain number of epochs, or underfitting, which would show as a stagnation of the loss at a higher value. While no validation curve is shown here, the consistent decrease in loss indicates that the model's optimization process (*Adam*) is working effectively, with no issues related to learning rate or instability in parameter updates. In conclusion, the model appears to have learned well over 100 epochs, minimizing the loss effectively. To further confirm its performance, it would be important to evaluate it on a separate validation or test set to ensure it generalizes well to unseen data.

The performance of the Artificial Neural Network (ANN) model was evaluated using the F1-score, a metric that balances precision and recall, providing a more comprehensive measure of model accuracy, especially in imbalanced datasets. The F1-score is the harmonic mean of precision and recall, and is calculated using the following formula:

$$F1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

where the precision is the ratio of correctly predicted positive observations to the total predicted positives:

$$\text{Precision} = \frac{TP}{TP + FP}$$

with TP being the number of true positives and FP the number of false positives. and where the recall is the ratio of correctly predicted positive observations to all observations in the actual class:

$$\text{Recall} = \frac{TP}{TP + FN}$$

with FN being the number of false negatives. The F1-score ranges from 0 to 1, with 1 indicating perfect performance (both precision and recall are perfect) and 0 indicating poor performance.

For the training dataset, the model achieved an accuracy of 77.5%, with an F1-score of 82.92%. This indicates that the model performed well in terms of both precision and recall during the training phase. For the test dataset, the accuracy slightly decreased to 75.5%, but the F1-score remained strong at 80.93%. This suggests that the model maintained its ability to make accurate predictions on unseen data, albeit with a minor decrease in overall accuracy.

6.1.3 XGBoost Model

In this study, we utilized the *XGBClassifier* package in *PyTorch* to implement the XGBoost model, allowing for efficient training and optimization. XGBoost, known for its gradient boosting approach, was applied alongside an Artificial Neural Network (ANN) to compare their predictive performance. The evaluation of the XGBoost model showed that it performed similarly to the ANN. On the training dataset, XGBoost achieved an accuracy of 78.3% with an F1-score of 79.62%, while on the test dataset, it obtained an accuracy of 73.5% and an F1-score of 81%. These results indicate that both models generalize well, with slight variations in their predictive capabilities. The purpose of comparing XGBoost with ANN was to determine which model performs better in this specific context. Based on the evaluation metrics, we observe that both models yield comparable results, demonstrating similar effectiveness in classification tasks. However, accuracy alone does not provide sufficient insights into how these models make predictions.

To gain a deeper understanding, the next step involves applying SHAP (SHapley Additive Explanations) and ALE (Accumulated Local Effects) to interpret the models. These explainability techniques will help uncover potential differences in how ANN and XGBoost derive their predictions, shedding light on feature importance and decision-making processes.

6.2 Explainability of the Model Output

As predictive models become more advanced, understanding their decision-making processes is essential for evaluating their reliability and improving their effectiveness. Interpretable models help ensure fairness, detect biases, and enhance performance by revealing how input features influence predictions. In the following sections, two different types of explainability are presented: feature importance, which quantifies the impact of individual variables, and visual explanations, which provide intuitive, graphical insights into model behavior.

6.2.1 Feature Importance

Feature importance helps identify which input variables have the most influence on a model's predictions. Understanding these contributions allows for better optimiza-

tion, interpretability, and trust in data-driven decision-making. In the following, several SHAP plots will be presented, illustrating their application to the trained ANN and XGBoost model to gain insight on the biggest drivers of a model's predictions.

SHAP for ANN Model

To interpret the predictions which we produced with an ANN model, we utilized the *DeepExplainer* from the SHAP (SHapley Additive exPlanations) library in *PyTorch*. The DeepExplainer method is specifically designed for deep learning models, including ANNs, and provides a comprehensive framework for understanding how input features influence the model's output. This tool leverages Shapley values, a game-theoretic approach, to attribute the contribution of each feature to the prediction, offering a more transparent and interpretable understanding of the model's decision-making process.

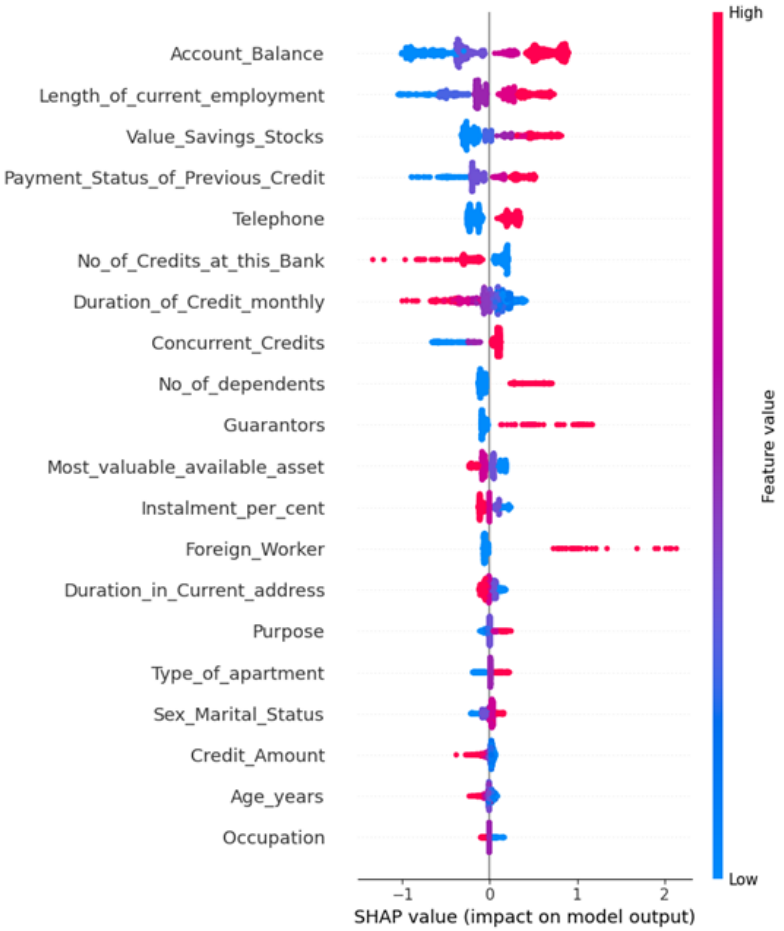

Figure 8: Summary SHAP Plot for the ANN model: This figure shows the importance of each feature, the most important feature is listed on the top.

The *DeepExplainer* operates by using the gradients of the model to approximate the contribution of each feature. This approach is well-suited for complex models like ANNs, where the relationships between input features and outputs can be highly non-linear and difficult to decipher. By integrating the *DeepExplainer* with our trained ANN, we were able to generate detailed visualizations, such as SHAP summary plots, which highlighted the relative importance of each feature and provided insights into how features interacted to influence predictions.

Fig. 8 shows us the Summary SHAP Plot. This global analysis tool (global meaning across all observations) helps us to identify which features are overall the strongest drivers for the predictions, whether positive or negative. The summary plot visualizes the contribution of each feature to each prediction. In the left-hand column, all features are listed, with the most important feature at the top. In this case the account balance of the borrower is the strongest driver, followed by the length of the current employment, savings stocks' value and so on. The weakest driver is the occupation of the borrower. The x-axis shows the SHAP values, which measure the impact on the prediction. Each point in the plot represents an observation in the dataset and shows whether the corresponding feature had a positive or negative impact on the prediction. The color of the dots indicates whether a characteristic has a high or low value for this observation. Red dots represent a high value and blue dots a low value for the individual characteristics. Fig. 8 shows that high account balances increase the prediction and thus have a positive influence on the prediction, whereas low account balances decrease the prediction and thus have a negative influence on the prediction. This also applies to the features *Length_of_current_employment*, *Value_Savings_Stock*, *Payment_Status_of_Previous_Credit* and *Telephone*. On the other hand, a low number of credits that the borrower holds at this bank has a positive effect on the forecast and a high number of credits has a negative effect. The density of the points along the x-axis illustrates how often a certain SHAP value occurs. A higher density indicates that many observations have a similar influence of the feature on the prediction. The vertical dispersion of a feature shows that its influence varies, possibly due to interactions with other features.

Having examined the overall importance of each feature across all observations, we now focus on the feature importance for a specific individual observation. To explore this, we utilize the Waterfall SHAP Plot, which decomposes each prediction into contributions from individual features. At the foundation of the SHAP waterfall plot is the base value, which represents the average model output across all observations and serves as the starting point for the explanation. Each feature is displayed as a bar, showing whether it pushes the prediction higher or lower. Features with a positive contribution shift the prediction to the right, while those with a negative contribution shift it to the left. The final model prediction for a given observation is obtained by summing the base value and the corresponding SHAP values.

To illustrate this process, consider a specific observation (Fig. 9). The values of each feature are shown on the y-axis and the base value is 0.408 and it has a model output of -1.585. By examining the SHAP values, we can trace how the model's prediction is derived from the base value. Insights from the SHAP summary plot indicate that a low account balance tends to push the prediction higher, which is reflected in the breakdown of contributions for this particular case.

A key question that arises is the interpretation of the model output. Since this is a classification problem with a binary target variable (0 or 1), one might expect the model output to be either 0 or 1. However, the value -1.585 represents the raw model output before conversion into a classification label. This raw value is passed through a sigmoid function (equation 28), transforming it into a probability estimate. In this case, the model assigns approximately 17% probability to receiving credit. A predefined threshold—0.5 in this instance—is then applied to determine the final

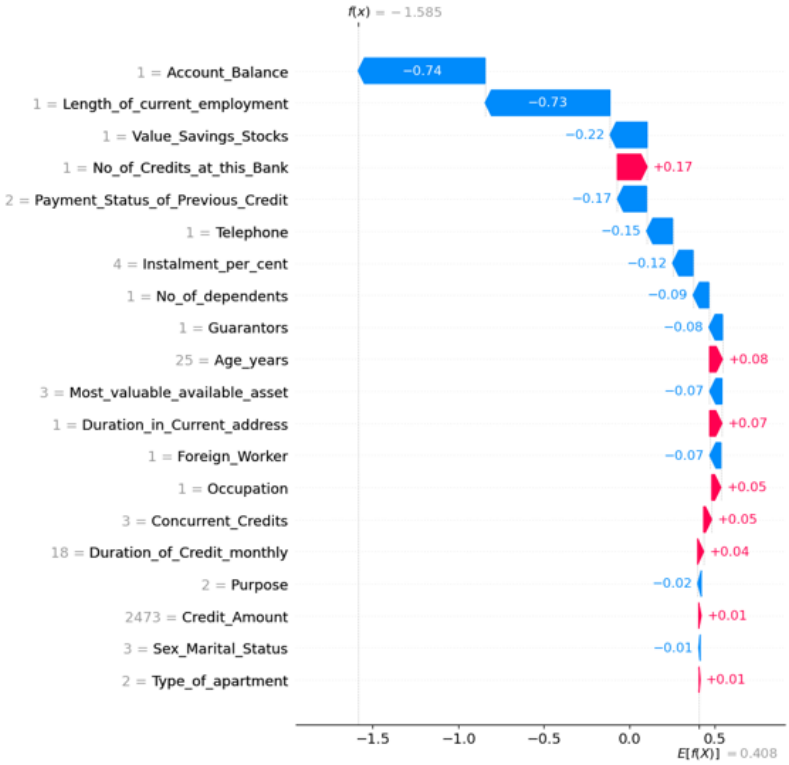

Figure 9: The Waterfall SHAP Plot showcasing each feature's contribution to the model prediction of the ANN model for a single observation.

classification. Since the predicted probability is below 0.5, the final prediction is 0 (no credit received).

$$\text{Probability} = \frac{1}{1 + e^{-(\text{model output})}} = \frac{1}{1 + e^{-(-1.585)}} = 0.17. \tag{28}$$

Ultimately, the SHAP waterfall plot provides a detailed breakdown of how each feature influences the model's decision, offering a transparent and interpretable view of the underlying decision-making process.

SHAP for XGBoost Model

In this study, we utilized the *TreeExplainer* from the *shap*-library in *PyTorch*, a specialized explainer tailored for tree-based models such as XGBoost. TreeExplainer is particularly well-suited for XGBoost because it leverages the tree structure of the model to compute exact SHAP values efficiently. This efficiency arises from the use of a fast algorithm that integrates seamlessly with the split-based decision paths in tree models, providing precise and computationally inexpensive feature attributions. By using TreeExplainer, we were able to quantify the contribution of each feature to the model's predictions at both the dataset and individual observation levels. This approach allowed us to identify the most influential features driving the model's decisions and to uncover patterns and relationships within the data that might otherwise remain hidden in a traditional black-box model analysis. Through the combination of XGBoost and TreeExplainer, we achieved a high-performing model with interpretable outputs, bridging the gap between accuracy and explainability. This interpretability is essential for validating the model's predictions and ensuring they align with domain knowledge and expectations.

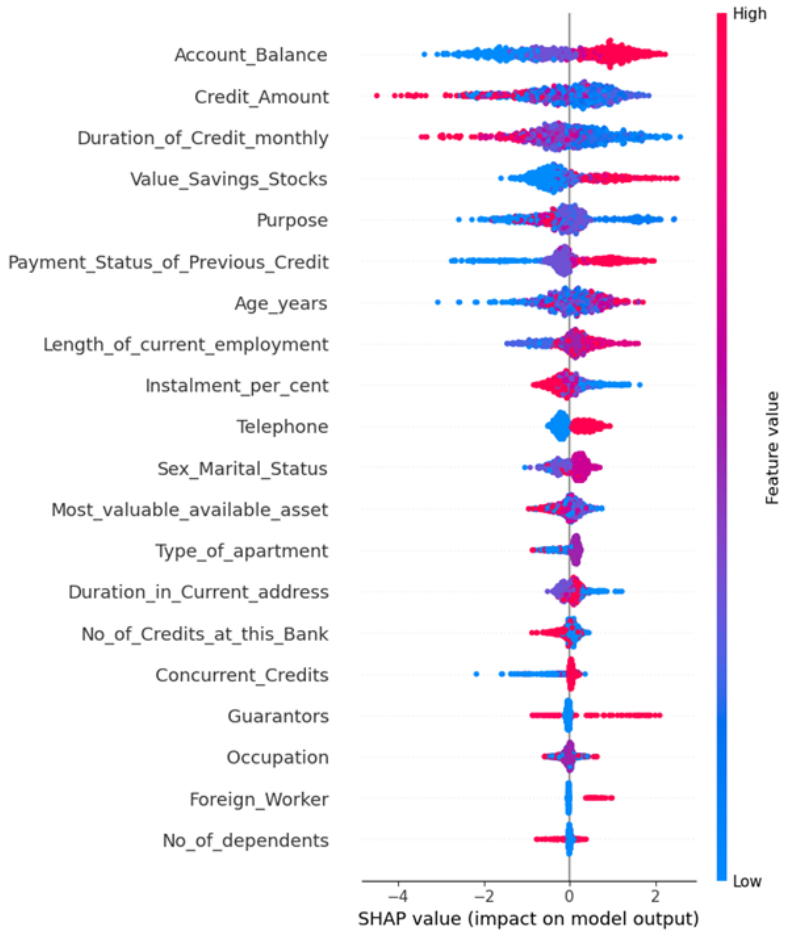

Figure 10: The Summary SHAP plot for the XGBoost model's predictions.

To gain a comprehensive overview of feature importance across the dataset, we utilized the Summary plot. Figure 10 provided a clear and concise depiction of how each feature influenced the predictions, ranked by their overall impact. The analysis revealed that the account balance of the borrower is the most critical feature in the model. Higher account balance values positively contribute to the predictions, signaling a stronger likelihood of creditworthiness, while lower values significantly weaken the prediction. Following this, the credit amount and the duration of the credit (in months) ranked as the second and third most important features, respectively. Interestingly, these features had an inverse relationship with the model's output: lower values for both the credit amount and the duration of credit contributed positively to the predictions, enhancing the likelihood of a favorable outcome. At the other end of the spectrum, the features with the least impact were occupation, foreign worker, and number of dependents. These variables showed minimal influence on the model's predictions, indicating that while they were included in the model, their contribution to the overall decision-making process was negligible compared to the other features.

By using the SHAP summary plot, we were able to clearly identify the most and least impactful features, providing valuable insights into the factors driving the model's decisions and highlighting areas where data or feature engineering might have a greater or lesser influence on model performance. But there are also some big differences to the ANN model regarding the feature importance. The XGBoost

model scores the credit amount of the borrower as the second most impactful out of all features which is very different from the ANN model, which ranked credit amount as the third least important feature. This illustrates the different training methods used in the various machine learning models such as ANN and XGBoost, resulting in identifying different patterns and structures within the data, which ultimately produce different results.

6.2.2 Visual Explainability

Visual explanation offers a clear and intuitive way to show how individual features influence a machine learning model's predictions. By analyzing ALE plots, we can gain insights into which features drive predictions and how their effects change across different values, leading to a more transparent understanding of the model's decision-making process.

In this study, we converted the output variable from type of *PyTorch* to *NumPy array* and utilized the ALE *explainer* from the *alibi.explainers* package to illustrate the ALE plots for 20 individual features for ANN and XGBoost, providing insights into how each feature influences the prediction results. In the ALE plot, the values of all features are already scaled through the preprocessing process.

ALE for ANN

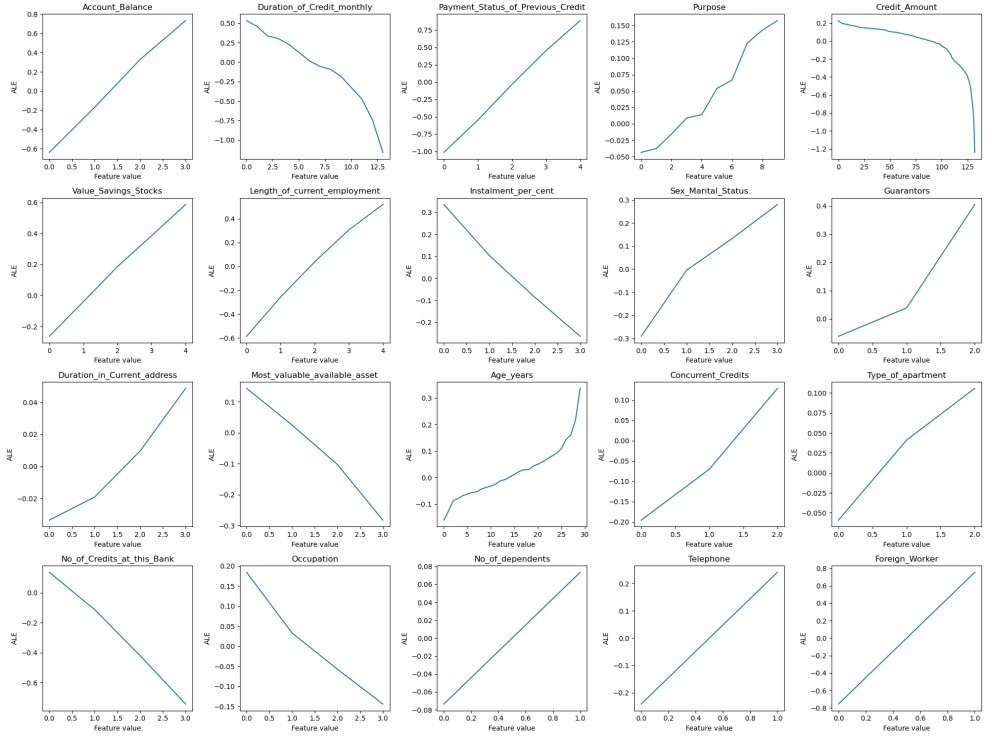

Figure 11: ALE Plots for ANN.

According to Figure 11, all features influence the prediction results of the ANN model, as evidenced by the ALE values that exhibit either linear or nonlinear (curved) patterns. To further analyze this influence, we focus on three features: *Age_years*, *Account_Balance*, and *Credit_Amount*, as shown in Figure 11. For the *Age_years* feature, the ALE plot reveals that younger individuals have a higher risk of credit default, as indicated by negative ALE values, while middle-aged and older individuals show a lower risk. This pattern may be attributed to the greater financial stability often associated with increased age. Regarding the *Account_Balance*

feature, its range spans from 1 to 4, and the ALE values indicate that individuals with an account balance greater than 2 have a significantly reduced risk of default. This suggests that higher account balances are strongly correlated with financial reliability. For *Credit_Amount*, the relationship is intuitive: as the credit amount increases, the ALE values rise, indicating a higher risk of default. This finding aligns with the understanding that larger loan amounts represent a greater financial burden, increasing the likelihood of default.

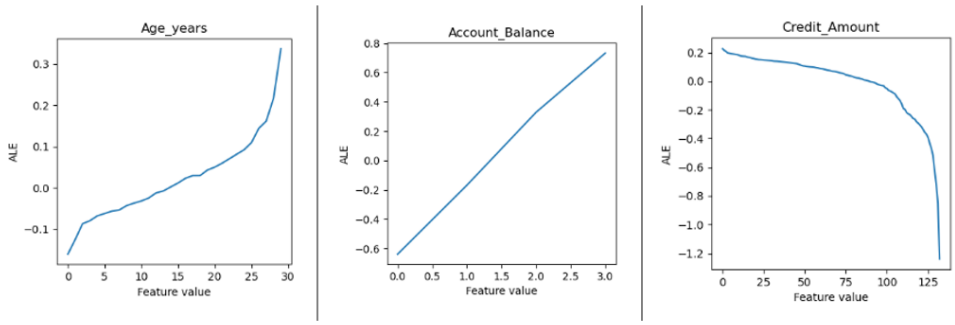

Figure 12: ALE plots for ANN for the features *Age_years*, *Account_Balance*, and *Credit_Amount*.

ALE for XGBoost

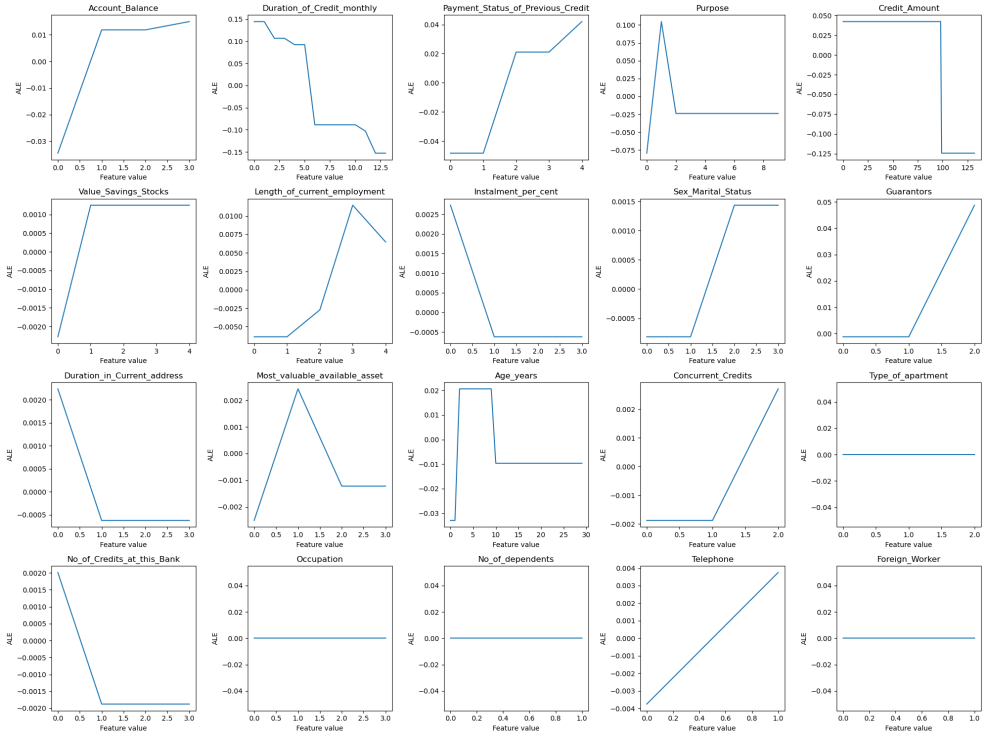

Figure 13: ALE Plots for XGBoost.

For the XGBoost model, the ALE (Accumulated Local Effects) plots 14 reveal that not all features significantly influence the prediction. Notably, the features *No_of_dependents*, *Type_of_apartment*, and *Occupation* exhibit minimal or no discernible impact, as their ALE values remain constant across feature values. This suggests that these features may not contribute meaningfully to the model's decisions. Additionally, the shapes of the ALE curves are generally not smooth but exhibit abrupt

changes, appearing more folded or piecewise linear. This indicates that the relationships between certain features and the model's predictions are not continuous but instead reflect sharp thresholds or categorical dependencies. Features such as *Account_Balance* and *Payment_Status_of_Previous_Credit*, however, demonstrate more significant and interpretable effects, showing notable changes in ALE values with varying feature values.

The ALE plots for XGBoost show that three features—*Age_years*, *Account_Balance*, and *Credit_Amount* demonstrate similar results to those observed for the ANN model. However, a slight difference is observed in the *Age_years* plot. In this case, middle-aged individuals show a slightly higher likelihood of taking credit compared to older individuals, as reflected by the ALE values. For *Account_Balance*, the trend remains consistent: individuals with a balance greater than 2 exhibit a significantly lower risk of default, reinforcing the strong correlation between higher account balances and financial stability. Similarly, the *Credit_Amount* feature continues to indicate that higher loan amounts correspond to a greater risk of default, a finding that aligns with expectations.

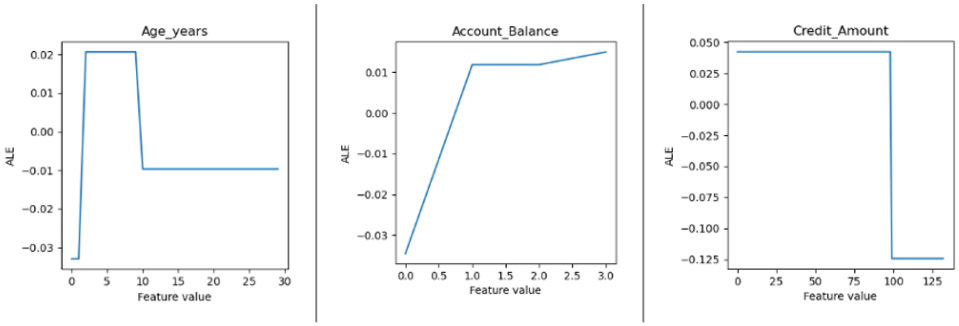

Figure 14: ALE plots for XGBoost for the features *Age_years*, *Account_Balance*, and *Credit_Amount*.

7 Discussion

7.1 Evaluation of the Results and Methodological Framework

The evaluation of the results obtained from the Artificial Neural Network (ANN) and XGBoost models provides insights into their predictive performance and effectiveness in assessing creditworthiness. Both models demonstrated comparable performance in terms of accuracy and F1-score, with ANN achieving an accuracy of 77.5% on the training set and 75.5% on the test set, while XGBoost attained 78.3% and 73.5%, respectively. The F1-scores further supported the similarity in performance, suggesting that both models are capable of identifying patterns in the dataset while maintaining generalizability.

The methodological framework employed in this study ensured a systematic and reproducible approach to model development and evaluation. The dataset underwent rigorous preprocessing, including normalization, standardization, and handling the imbalanced data, to optimize model performance. The training process incorporated hyperparameter tuning by grid search to enhance predictive accuracy while mitigating overfitting. Beyond performance metrics, model explainability was a central aspect of the evaluation. Post hoc explainability techniques, including SHAP (Shapley Additive Explanations) and ALE (Accumulated Local Effects), were applied to interpret model predictions and analyze feature importance. These methods provided insights into the underlying decision-making processes, addressing the black-box nature of machine learning models. By leveraging explainability techniques, we identified key factors influencing creditworthiness predictions. Both ANN and XGBoost assigned significant importance to variables such as account balance and credit amount, though their specific decision boundaries differed. The comparison of SHAP and ALE results highlighted variations in how each model processes and utilizes input features, underscoring the importance of employing multiple interpretability methods for a holistic understanding of machine learning models.

The findings of this study contribute to the growing field of AI-driven credit assessment by demonstrating the potential of ANN and XGBoost in predictive modeling while emphasizing the need for transparency in financial decision-making. Future research may explore the integration of additional explainability techniques or alternative modeling approaches to enhance model interpretability and reliability further.

7.2 Strengths and Weaknesses of the Methods

Artificial Neural Networks (ANN) offer high predictive power and the ability to model complex, non-linear relationships in data, making them well-suited for creditworthiness prediction. However, they require large amounts of data for effective training and are computationally intensive, especially with deep architectures. Additionally, their black-box nature makes interpretation difficult, necessitating post hoc explainability methods to enhance transparency.

SHAP provides a comprehensive measure of feature importance by fairly distributing contributions based on game theory, making it valuable for both global and local interpretability. However, it is computationally expensive, particularly for models

with many features, and can be difficult to apply efficiently to large datasets. ALE, on the other hand, offers a more computationally efficient approach by focusing on the marginal effects of features and it also addresses the issue that Partial Dependence Plots assume feature independence, which can lead to misleading interpretations when features are correlated. But ALE only provides global explanations, limiting insights into individual predictions. Both SHAP and ALE help address the black-box problem of machine learning models by improving interpretability and offering insights into how features influence predictions. While their methods differ, combining SHAP's detailed attribution with ALE's computational efficiency provides a more comprehensive understanding of model behavior.

8 Conclusion

In this study, we explored the use of Artificial Neural Networks (ANN) and XGBoost for predicting creditworthiness, comparing their performance and interpretability. We applied machine learning techniques to analyze the *german credit* dataset, preprocessing the data and optimizing model parameters to achieve accurate and reliable predictions. Both ANN and XGBoost demonstrated comparable performance based on evaluation metrics such as accuracy and F1-score, highlighting their effectiveness in credit risk assessment. Beyond performance evaluation, we focused on model interpretability by utilizing post hoc explainability methods. SHAP (SHapley Additive Explanations) and ALE (Accumulated Local Effects) were employed to address the black-box nature of these models, providing insights into feature importance and decision-making processes. While SHAP allowed for both global and local explanations, ALE offered a computationally efficient way to analyze feature effects, ensuring a transparent and interpretable model evaluation. Our findings underscore the importance of integrating explainability techniques in AI-driven financial decision-making to enhance trust, accountability, and regulatory compliance. Future research could explore alternative modeling techniques, additional explainability methods, or hybrid approaches to further improve the accuracy and interpretability of creditworthiness predictions.

Literature

[1] Ravi Abhinav. *Improving Class Imbalance with Class Weights in Machine Learning*. 2023. URL: https://medium.com/@ravi.abhinav4/improving-class-imbalance-with-class-weights-in-machine-learning-af072fdd4aa4 (visited on 02/08/2025).

[2] Daniel W Apley and Jingyu Zhu. "Visualizing the effects of predictor variables in black box supervised learning models". In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 82.4 (2020), pp. 1059–1086.

[3] Victor Chang et al. "Prediction of bank credit worthiness through credit risk analysis: an explainable machine learning study". In: *Annals of Operations Research* (2024), pp. 1–25.

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

[5] Ian Covert and Su-In Lee. "Improving kernelshap: Practical shapley value estimation via linear regression". In: *arXiv preprint arXiv:2012.01536* (2020).

[6] Christian W Dawson and Robert Wilby. "An artificial neural network approach to rainfall-runoff modelling". In: *Hydrological Sciences Journal* 43.1 (1998), pp. 47–66.

[7] Mohamad Forouzanfar et al. "Comparison of feed-forward neural network training algorithms for oscillometric blood pressure estimation". In: *4th International Workshop on Soft Computing Applications*. IEEE. 2010, pp. 119–123.

[8] Leonida Gianfagna and Antonio Di Cecco. *Explainable AI with python*. Vol. 4. Springer, 2021.

[9] Ian Goodfellow. *Deep learning*. Vol. 196. MIT press, 2016.

[10] Uday Kamath and John Liu. *Explainable artificial intelligence: an introduction to interpretable machine learning*. Vol. 2. Springer, 2021.

[11] Robert Kwiatkowski. *Gradient Descent Algorithm — a deep dive*. 2021. URL: https://medium.com/towards-data-science/gradient-descent-algorithm-a-deep-dive-cf04e8115f21 (visited on 02/08/2025).

[12] Haiping Lu and Shuo Zhou. *Turing Course: An Introduction to Transparent Machine Learning*. 2022. URL: https://alan-turing-institute.github.io/Intro-to-transparent-ML-course/index.html (visited on 02/08/2025).

[13] Scott Lundberg. "A unified approach to interpreting model predictions". In: *arXiv preprint arXiv:1705.07874* (2017).

[14] Scott Lundberg and Su-In Lee. "An unexpected unity among methods for interpreting model predictions". In: *arXiv preprint arXiv:1611.07478* (2016).

[15] Scott M Lundberg et al. "Explainable AI for trees: From local explanations to global understanding". In: *arXiv preprint arXiv:1905.04610* (2019).

[16] Sujith Mangalathu, Seong-Hoon Hwang, and Jong-Su Jeon. "Failure mode and effects analysis of RC members based on machine-learning-based SHapley Additive exPlanations (SHAP) approach". In: *Engineering Structures* 219 (2020), p. 110927. ISSN: 0141-0296. DOI: https://doi.org/10.1016/j.engstruct.2020.110927. URL: https://www.sciencedirect.com/science/article/pii/S0141029620307513.

[17] Saumitra Mishra, Bob L Sturm, and Simon Dixon. "Local interpretable model-agnostic explanations for music content analysis." In: *ISMIR*. Vol. 53. 2017, pp. 537–543.

[18] Christoph Molnar. *Interpretable Machine Learning A Guide for Making Black Box Models Explainable*. 2020. URL: https://christophm.github.io/interpretable-ml-book/ (visited on 01/28/2025).

[19] Ekaterina V Orlova. "Methodology and models for individuals' creditworthiness management using digital footprint data and machine learning methods". In: *Mathematics* 9.15 (2021), p. 1820.

[20] Paola Sebastiani, MM Abad, and MF Ramoni. *The Data Mining and Knowledge Discovery Handbook*. 2005, pp. 193–230.

[21] Musa Segun, Victoria Yemi-Peters, and Sunday Eric Adewumi. "Credit worthiness prediction: Approaches and methods". In: *Ukrainian Journal of Educational Studies and Information Technology* 12.1 (2024), pp. 15–31.

[22] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. "Learning important features through propagating activation differences". In: *International conference on machine learning*. PMlR. 2017, pp. 3145–3153.

[23] Zalak L Thakker and Sanjay H Buch. "Effect of Feature Scaling Pre-processing Techniques on Machine Learning Algorithms to Predict Particulate Matter Concentration for Gandhinagar, Gujarat, India". In: ().

[24] Regina Esi Turkson, Edward Yeallakuor Baagyere, and Gideon Evans Wenya. "A machine learning approach for predicting bank credit worthiness". In: *2016 Third International Conference on Artificial Intelligence and Pattern Recognition (AIPR)*. IEEE. 2016, pp. 1–7.

[25] Jilei Yang. "Fast treeshap: Accelerating shap value computation for trees". In: *arXiv preprint arXiv:2109.09847* (2021).

List of Figures

[78] Molnar 2020.
[79] Molnar 2020.
[80] Molnar 2020.

List of Abbreviations

AI – Artificial Intelligence

ANN – Artificial Neural Network

ALE – Acumulated Local Effects

M-Plot – Marginal Plot

ML – Machine Learning

PDP – Partial Dependence Plot

ReLU – Rectified Linear Unit

SHAP – SHapley Additive exPlanations

XAI – Explainable Artificial Intelligence

XGBoost – Extreme Gradient Boosting