Disentangling the AI Black-Box Model: From Direct to Indirect Influence

Serly Ishkhanian

A Thesis in
in
The Department of
of
Mathematics and Statistics

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Arts (Mathematics) at
Concordia University
Montreal, Quebec, Canada

June, 2025

©Serly Ishkhanian, 2025

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared
By: Serly Ishkhanian
Entitled: Disentangling the AI Black-Box Model: From Direct to Indirect Influence
and submitted in partial fulfillment of the requirements for the degree of

Master of Arts (Mathematics)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.
Signed by the final Examining Committee:

_____ Chair
Dr. Marco Bertola

_____ Thesis Supervisor
Dr. Simone Brugiapaglia

_____ Thesis Co-Supervisor
Dr. Weiqi Wang

_____ Examiner
Dr. Arusharka Sen

Approved by

_____ Dr. Lea Popovic, Graduate Program Director

_____ Dr. Pascale Sicotte, Dean of Faculty of Arts and Science

Date: June 30, 2025

Abstract

Disentangling the AI Black-Box Model: From Direct to Indirect Influence

Serly Ishkhanian

Due to the widespread use of artificial intelligence and machine learning models across numerous domains that directly impact individuals' lives, it is essential to ensure that these models are making their decisions in a non-discriminatory manner. That is, biases encoded in the underlying training data are not reflected in the output of the model. This is, however, often hard to control since the predictive accuracy of these models come at the cost of interpretability.

The purpose of this thesis is to investigate and apply methods for interpreting black-box machine learning models to bring more transparency into their decision-making process by analyzing both the direct and indirect influence of input features on model predictions. We begin by studying the theoretical background of the SHAP (SHapley Additive exPlanations) framework, Shapley values, and their implementation in measuring direct influence. We explore the SHAP Python package for both local and global explanations and visualization of direct feature influence on both synthetic and real-world datasets. We then transition to studying indirect influence using the Disentangled Influence Audit procedure, which uses adversarial training to learn disentangled latent representations for auditing hidden dependencies. After presenting all the background information, we implement this procedure and evaluate these methods through numerical experiments on synthetic functions and real-world datasets including the Adult Income and Montréal Housing datasets. In addition to these experiments, we also examine how "data-hungry" these auditing methods are by examining their convergence as a function of the number of samples required. Our results demonstrate the importance of auditing both direct and indirect pathways of influence to promote interpretability and fairness in complex models.

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Dr. Simone Brugiapaglia, and my co-supervisor, Dr. Weiqi Wang, for their invaluable guidance and support throughout this thesis. Their expertise, patience, and encouragement have been instrumental in shaping this work.

I am also grateful to Dr. Brugiapaglia for giving me the opportunity to join the PropTech research team, which led to the development of this thesis. Through this experience, I had the privilege of collaborating with a team of professors and researchers, including Dr. Alessandra Renzi (Concordia), and Dr. Tamara Vukov (UdeM), who were the co-principal investigators of the project. We also had the opportunity to present our work at Concordia's Applied AI Institute.

I would also like to thank Dr. Arusharka Sen for taking part in the Examining Committee

Finally, I would like to thank my family and significant other for their unwavering support and encouragement, without them none of this would have been possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter introduces the motivation for this research, driven by the increasing dependence on black-box machine learning models and the need for transparency in their decision-making processes. We begin by discussing recent literature on algorithmic bias and the challenges of interpreting complex models. We also present methods that aim to uncover direct and indirect feature influences with their challenges. Finally, we outline the objectives of the thesis and provide an overview of the structure.

1.1 Motivation and Literature Review

In recent years, there has been a substantial increase in the widespread employment and improvement of Artificial Intelligence (AI) and Machine Learning (ML) systems, where they are taking over decision-making processes, having direct effect on individual's life. These complex models have become prominent in numerous domains such as in healthcare, insurance, hiring employees, criminal justice, finance, and even housing allocation due to their predictive precision. However, this accuracy often comes at a cost; transparency and interpretability [12].

This lack of transparency and interpretability has raised a lot of concern in both research and practice, as it is more difficult to assess the decisions made by these models and whether they were justifiable and fair. Numerous studies have revealed that ML models can encode bias present in their training data [7]. Obeymer et al. [22] revealed that a commonly used healthcare algorithm exhibited racial bias, where it classified black patients at the same risk level as white patients were, when in reality, the black patients were more severely ill. Sensitive attributes such as race and gender can directly impact a model's output, however, even if they are omitted from the training data set during the model's training process, the effects of these attributes can still remain in the data through other variables (proxy variables). For instance, a common proxy variable for race is "ZIP codes" in datasets with segregated regions demographic regions [1]. For Obeymer et al. the feature "health costs" was the proxy variable for "health needs," reflecting the historical patterns where black patients had lower healthcare spending. Similarly, the ProPublica's investigation showed that algorithms used for criminal risk assessment in courtrooms demonstrated racial bias, where

black defendants were twice as likely to be falsely classified as high risk compared to white defendants [13]. This also shows how features indirectly affect model decisions even when explicit demographic information is not included in the data.

When these models have great influence on people's lives, it is of great importance to ensure that the predicted outputs do not contain discriminatory effects. To understand how input features are affecting the model's output there has been extensive research in understanding and measuring the direct influence of features on model's predictions such as SHAP (SHapely Additive eXplanations) values [19], LIME (Local Interpretable Model-Agnostic Explanations) [23], DeepLift [25]. These tools are commonly utilized in the field for auditing models, explaining predictions, and debugging. Despite their benefits, these methods share a common issue: they are mostly used for identifying which features directly influence black-box models. They estimate the impact of a feature by analyzing what happens when the feature itself is perturbed or removed. However, as discussed, even if a feature is explicitly excluded from being directly used by the model, its effect can influence the predictions through proxy variables.

A growing area of research has explored different methods of auditing black-box models to address these issues. One technique is the Black-box Auditing (BBA) method by Adler et al. [1], which evaluates indirect influence of a feature by obscuring the data, as minimally as possible to minimize information loss, until the feature of interest cannot be predicted from the remaining features, then measuring how this modification changes the model's output. BBA does not require access to the internal structure of the black-box model being audited as some other methods do. However, BBA has some limitations; BBA handles only numerical and categorical data, and the obscuring process applied is dependent on the data type. Additionally, this method only provides a global view of the indirect influence of a feature, offering no insight at the individual level.

To address these limitations, Marx et al. [21] propose a novel approach, the Disentangled Influence Audit method, that uses adversarial training to learn a lower-dimensional representation independent of protected features. Unlike BBA, their approach can be applied to categorical, numerical, and image data. Moreover, this approach also provides not only global interpretation of the indirect influence of features, but also how the features contribute in the decision making process of a specific instance.

1.2 Objective and Contributions

The objective of this thesis is to dive deeper into black-box models, and analyze both direct and indirect feature influences on model predictions. To evaluate direct influence, the primary methodology we employ in this thesis is SHAP values introduced by Lundberg and Lee [19]. To analyze indirect influence, we study and apply the Disentangled Influence Audit method by Marx et al. to audit indirect influences. We will observe how the combination of direct and indirect influence methods enables us to gain insights on how features affect the decision-making process of models.

This thesis provides the following three contributions. First, we perform a series of experiments using SHAP values to visualize and analyze the direct feature influence in different applications, including synthetic datasets, the Adult Income dataset, and the Montréal Housing Dataset, hence providing insight into how features affect model predictions. Second, we conduct a convergence analysis of SHAP on synthetic data to evaluate the convergence behavior of SHAP estimates to changes in sample size. Lastly, we implement the Disentangled Influence Audit (DIA) method introduced by Marx et al. [21] to identify indirect feature influence. To achieve this, we developed the model from scratch, and validated it by reproducing two key experiments from the original paper. We also extend the application of this method to the Montréal Housing Dataset to detect indirect influences and proxy variables that are not captured by direct influence methods.

Now, we outline the structure of this thesis. In Chapter 2, we focus on direct influence. We introduce the theoretical foundation of SHAP values, which originate from cooperative game theory, and explain how they are used to interpret model predictions. To demonstrate the practical application of SHAP, we conduct a series of experiments on synthetic and real-world datasets. This includes reproducing two key experiments from [21], and we extend our analysis of the method to the Montréal Housing Dataset. Additionally, we perform a convergence analysis of the SHAP method on the synthetic data to assess its reliability and dependence on data.

In Chapter 3, we look into indirect influence. After reviewing necessary background information, such as the use of Adversarial Autoencoders to apply the Disentangled Influence Audit method, we provide an in-depth explanation of the implementation of this method. This method allows us to detect proxy variables and uncover their impact on model outputs, which are not sensed by direct influence methods. Lastly, we discuss the results we obtain from applying this method on the same experiments as in Chapter 2.

Finally, we summarize the main findings in Chapter 4 and reflect on how the insights gained from both direct and indirect influence analysis can aid in auditing black-box models. We also discuss potential directions for future work and broader applications of these methods. Additional implementation details, and further results are provided in the Appendix.

Chapter 2

Measuring Direct Influence

Suppose that we have a model that predicts the annual bonus that an employee will receive based on the employee's age, degree, performance, experience and the number of days they were late. One would be interested in understanding the process of how the model makes its decisions and how the features contribute to the prediction of the model. Although knowing the importance of a feature is important, it does not answer the previous questions as it does not provide any information on how much each feature contributed to the decision-making process. Assume that you have two employees of the same age, the same education, and the same performance level, but one received a higher bonus than the other. How did the model make this decision? Was it fair? To answer these questions, Lundberg and Lee [19] introduced SHAP (Shapley Additive exPlanations) values, a unified approach for interpreting model predictions and understanding how a feature influences those predictions. SHAP is based on a game-theoretic approach to explain the output of any ML model. Hence, to better understand what SHAP values are, it is essential to first explore its theoretical foundation, Shapley values.

In this chapter, we present the fundamental concepts, notations, and tools that will be used throughout this thesis. We begin by presenting the foundational idea of Shapley values, outlining their key properties, and the method of computation. In Section 2.2, we provide an overview of various techniques for interpreting complex machine learning models. Then, in Section 2.3 we discuss the primary interpretability tool used in this thesis, SHAP. We also explain how to interpret the visual outputs generated by the SHAP python package, before concluding with a presentation of the numerical experiments in which SHAP was applied to measure direct influence of features.

2.1 Shapley Values

To gain an understanding of what Shapley values are, let us start with a simple example. Suppose that there is a team of three players who wins 1^{st} place in the game they are participating in, earning a prize of $10,000. Should the prize be distributed equally among the members of the team? While equally diving the prize is easy, would it be fair if they all have not contributed equally? To address this issue, Lloyd Shapley introduced Shapley

value in 1951 [24]. It is a powerful method in game theory to fairly distribute the total gains of a game among its players depending on each player's Marginal Contribution (MC) to every possible coalition of players. Over the years, the impact of the Shapley value has extended to various fields from economics to machine learning, and political science, because it provides a way of understanding how allocations are made. In this section, we define what "fair" means and explain how Shapley values are calculated.

2.1.1 Notation

Let N be the set of all players, where $|N| = p$. Consider a pre-defined set function $v : 2^N \to \mathbb{R}$, which assigns a real-valued payoff to every possible coalition $S \subseteq N$. For any coalition S excluding the i-th player, (i.e., $S \subseteq N \setminus \{i\}$ where $i \in N$), the value (or payoff) of coalition S is denoted by $v(S)$. The value generated when all members work together (i.e., total value of the game) is denoted by $v(N)$, and the Marginal Contribution of Player i (MC_i) to coalition S is given by $v(S \cup \{i\}) - v(S)$. Lastly, the Shapley value of Player i is denoted as $\phi_i(N, v)$.

2.1.2 Properties

The fairness of the Shapley value is defined through the following axioms [24]:

Property 1: Efficiency The efficiency axiom states that the sum of each player's contribution (i.e., the sum of the Shapley values) must equal the total value generated by the grand coalition:

$$v(N) = \sum_{i=1}^{p} \phi_i(N, v).$$

Property 2: Symmetry If two players i and j contribute equally to every coalition, then they should receive the same Shapley value:

$$\text{If } v(S \cup \{i\}) = v(S \cup \{j\}) \text{ for all } S \subseteq N \setminus \{i, j\}, \text{ then } \phi_i(N, v) = \phi_j(N, v).$$

Property 3: Dummy If a player does not increase the value of any coalition they join, then their Shapley value is zero:

$$\forall S \subseteq N \setminus \{i\}, \text{ if } v(S \cup \{i\}) = v(S), \text{ then } \phi_i(N, v) = 0.$$

Property 4: Additivity If the value function is the sum of two separate games $v = v_1 + v_2$, then the Shapley value is the sum of the individual Shapley values:

$$\phi_i(N, v_1 + v_2) = \phi_i(N, v_1) + \phi_i(N, v_2).$$

2.1.3 Calculation of Shapley Values

The Shapley value of Player i (ϕ_i) in a p-player game is given by [24]:

$$\phi_i = \sum_{S \subseteq \{1,\ldots,p\} \setminus \{i\}} \frac{|S|! \cdot (p - |S| - 1)!}{p!} \underbrace{(\text{val}(S \cup \{i\}) - \text{val}(S))}_{\text{MC of Player } i},$$

where:

- $|S|$ is the number of players in coalition S excluding Player i.
- $p!$ is the number of ways to form a coalition of p players.
- $|S|!$ is the number of ways to form a coalition of $|S|$ players before player i joins.
- $(p - |S| - 1)!$ is the number of ways players can join after Player i joins.

To better understand the calculation, return to the example of a team of three players who won a prize of \$10,000. How should the prize be distributed?

Every player is unique with different strengths, meaning that under different scenarios, such as if they were to play alone or in a different group setting, maybe they would not win first place. For instance, when Player 1 and Player 2 play alone, they win third place, but when teamed together, their combined skillset enables them to win second place. Assume the conditions of the game are as follows: First place winner gets a prize of \$10,000, second place receives \$8,000, and third place receives \$6000. To explain how the contribution of each player is computed, consider all the following different possible scenarios and their corresponding rewards:

- If no player participates, the prize is \$0.
- If **Player 1** plays alone, the prize is \$6,000.
- If **Player 2** plays alone, the prize is \$6,000.
- If **Player 3** plays alone, the prize is \$0.
- If **Players 1 and 2** play together, they share a prize of \$8,000.
- If **Players 1 and 3** play together, they share a prize of \$8,000.
- If **Players 2 and 3** play together, they share a prize of \$6,000.
- If **all three players** participate together, they share a total prize of \$10,000.

Let i denote a player in the game. The Shapley value of Player i is the weighted sum of the marginal contributions of Player i:

$$\phi_i = \omega_1 \cdot MC_1 + \omega_2 \cdot MC_2 + \omega_3 \cdot MC_3 + \omega_4 \cdot MC_4, \tag{2.1}$$

where MC_k is the marginal contribution of the player i to the k-th coalition, and ω_k is the corresponding weight.

To compute the marginal contributions of Player 1, we need to calculate how much extra value Player 1 brings to the different teams. Specifically, we examine Player 1's contribution when joined to the following coalitions: 0 players, Player 2, Player 3, and Players 2 and 3. The marginal contribution in each case is calculated as the difference between the value of the coalition including Player 1 and the value of the same coalition without Player 1.

- **Marginal contribution of Player 1 to the coalition with no players:**

$$MC_1 = v(\{1\}) - v(\emptyset) = 6000 - 0 = 6000.$$

- **Marginal contribution of Player 1 to the coalition with Player 2:**

$$MC_2 = v(\{1, 2\}) - v(\{2\}) = 8000 - 6000 = 2000.$$

- **Marginal contribution of Player 1 to the coalition with Player 3:**

$$MC_3 = v(\{1, 3\}) - v(\{3\}) = 8000 - 0 = 8000.$$

- **Marginal contribution of Player 1 to the coalition with Players 2 and 3:**

$$MC_4 = v(\{1, 2, 3\}) - v(\{2, 3\}) = 10000 - 6000 = 4000.$$

These values will be used in formula 2.1 to compute Player 1's overall contribution. Now, how are the weights (ω_k's) calculated? The weights are the probabilities of the marginal contributions. It is the probability that Player 1 makes a marginal contribution to a certain coalition.

For example, let's calculate the probability that player 1 joins a group containing Players 2 and 3 (ω_4). There are 6 ways of having coalitions of 3 players,

123
132
213
312
231
321

Since we want the ones where Player 1 joins the group of Players 2 and 3, this implies that Player 1 should come last. From the above list, we have 2 cases where Player 1 joins a team already containing Players 2 and 3. Hence, the probability is:

$$\omega_4 = \frac{2}{6} = \frac{1}{3}.$$

This way, we obtain the following weights:

$$\omega_1 = \frac{1}{3}, \quad \omega_2 = \frac{1}{6}, \quad \omega_3 = \frac{1}{6}.$$

Hence, we can compute the Shapley value of Player 1 as follows:

$$\phi_1 = \frac{1}{3} \cdot 6000 + \frac{1}{6} \cdot 2000 + \frac{1}{6} \cdot 8000 + \frac{1}{3} \cdot 6000 = 5000.$$

Meaning, from the prize of \$10,000, Player 1 should receive \$5,000.

Following the same process, we could calculate how much Player 2 and Player 3 should receive:

$$\text{Shapley value of Player 2} = \frac{1}{3} \cdot 6000 + \frac{1}{6} \cdot 2000 + \frac{1}{6} \cdot 6000 + \frac{1}{3} \cdot 2000 = \$4,000.$$

$$\text{Shapley value of Player 3} = \frac{1}{3} \cdot 0 + \frac{1}{6} \cdot 2000 + \frac{1}{6} \cdot 0 + \frac{1}{3} \cdot 2000 = \$1,000.$$

Hence, the total prize of \$10,000 is fairly distributed among the three players, with Player 1 receiving \$5,000, Player 2 receiving \$4,000, and Player 3 receiving \$1,000. This example demonstrates how the Shapley value accounts for each player's contribution across all possible team combinations.

2.2 Interpreting Complex Models and the Rise of SHAP

In machine learning, interpretability is crucial, especially in addressing some of the current challenges related to fairness and accountability. Incidents such as biased and unexplainable decisions highlight the need to understand why a model makes certain predictions, as it can not only help build trust by providing transparency, but also aid in debugging. One of the simplest and easiest models to interpret is Linear Regression, which is a self-explanatory model since the model coefficients represent each feature's effect on the outcome. However, the simplicity of linear regression comes at the cost of flexibility; these models lack flexibility in capturing nonlinear patterns in more complex datasets [10]. Hence, as we move toward more complex models, such as deep neural networks [14], although they can be more accurate, they are much harder to interpret. These models require specialized post-hoc methods to explain their behavior.

Post-hoc interpretability methods analyze the model after training. These methods can be model-agnostic, or model-specific:

- **Model-agnostic methods** analyze how the model output changes with respect to changes in the feature inputs without considering the model structure. Examples of such model-agnostic methods include LIME [23], and SHAP [21]. These methods can be further divided into two groups:
  - Local methods explain individual predictions.

  - Global methods explain how features affect the predictions of the model on average (across the entire dataset).

- **Model-specific methods** analyze the model structure for a deeper understanding. Unlike model-agnostic methods, by the name, model-specific methods are applicable only to specific models, and they use a reverse engineering approach to provide explanations of how the specific models are making their predictions.

So, how can we explain the predictions made by complex machine learning models? Since these models are not self-explanatory, we need simpler explanation models to understand them. Lundberg and Lee define these explanation models as any interpretable approximation of the original function [19]. Let f be the original prediction model, and g be the explanation model used to explain a model's prediction $f(x)$ for a specific input x (for example, the input can be image data, bag-of-words, or tabular data). Explanation models usually use simplified binary input representations, denoted as z', where each entry of z' is either 1 or 0, representing the inclusion or exclusion of a feature in the explanation model, respectively. To simply put it, if x can be approximated with z' (such that $x \approx z'$), then we must have that $f(x) \approx g(z')$. Moreover, Lundberg and Lee provide the following definition, which ensures that these explanation models g express model predictions as a sum of contribution from each input features:

Definition 2.1. ***Additive feature attribution methods*** *have an explanation model that is a linear function of binary variables. The model is defined as with ϕ_0 being the average output of the model, and ϕ_i representing the feature effect:*

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i, \tag{2.2}$$

where $z' \in \{0,1\}^M$, M is the number of simplified inputs, and $\phi_i \in \mathbb{R}$.

It should be noted that the simplified inputs z' that these explanation models use are mapped back to the original input space using a mapping function $x = h_x(z')$. There are different mapping functions h_x depending on the input space. For tabular data, h_x maps the binary vector z' back to the original input space. For image data, where each feature cannot be considered as a feature, h_x treats the image as a set of super-pixels. It then maps 1 to include the original value of the super-pixel, and maps 0 in a certain format depending on the method (either they consider 0 as excluding it or replacing the value of the super-pixel with the average of its neighboring pixels) [23]. For instance, consider an input data that contains the following information of an individual: Age, Body Mass Index (BMI), and Gender, respectively.

$$\mathbf{x} = [25, 30, M],$$

indicating that this individual is 25 years old Male, with a BMI of 30. Out of all possible subsets, one possible binary vector could be,

$$\mathbf{z'} = [1, 0, 1],$$

where the value 1 would inform the explanation model to include the corresponding values of the features. Then, the mapping function h_x provides the following expression:

$$h_x(z') = [25, 0, M].$$

In the upcoming sections, we provide examples that would demonstrate how it is implemented during the calculation process of feature attributions.

Being able to approximate a model prediction $f(x)$ with the expression in Definition 2.1, it becomes easier to interpret the prediction since we can observe the contribution of each input feature through their corresponding coefficients, ϕ_i.

Lundberg and Lee show that there are several explanation methods that use the same explanation model as definition 2.1 to interpret complex models, where these methods represent a model output as an additive sum of the contributions from each feature, making the model's behavior more interpretable. The methods differ in their approach for computing feature attributions ϕ_i. Lundberg and Lee define this class of methods the Additive Feature Attribution method.

Furthermore, Lundberg and Lee state that any reliable explanation method should satisfy the following three properties:

- **Local Accuracy:** The explanation model output equals the original model output for the instance being explained.

$$f(x) = g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i.$$

- **Missingness:** Features that are absent (i.e. $z'_i = 0$) receive zero attribution ($\phi_i = 0$).
- **Consistency:** Let $f_x(z') = f(h_x(z'))$, and let $z' \backslash i$ denote setting $z'_i = 0$. For any two models f and f', if

$$f'_x(z') - f'_x(z' \backslash i) \geq f_x(z') - f_x(z' \backslash i) \quad \text{for all } z' \in \{0,1\}^M, \tag{2.3}$$

  then $\phi_i(f', x) \geq \phi_i(f, x)$. That is, if a model changes so that a feature's contribution to predictions increases or stays the same regardless of context, its attribution should not decrease.

These three properties are essential to ensure that feature attributions are meaningful and trustworthy. Specifically, Lundberg and Lee showed that there is only one explanation method that satisfies all three properties and adheres to the additive nature outlined in Definition 2.1 by proving the following theorem where the feature effects (ϕ_i) from the Definition are the Shapley values [19].

Theorem 2.1. *Only one possible explanation model g follows Definition 2.1 and satisfies Properties 1, 2, and 3:*

$$\phi_i(f,x) = \sum_{z' \subseteq x'} \frac{|z'|!\,(M-|z'|-1)!}{M!}\left[f_x(z') - f_x(z'\backslash i)\right], \tag{2.4}$$

where $|z'|$ is the number of nonzero entries in z' and $z' \subseteq x'$ represents all z' vectors where the nonzero entries are a subset of the nonzero entries in x.

In the literature, there are several explanation methods (under the class of Additive Feature Attribution methods) to interpret individual model predictions such as LIME [23], DeepLIFT [25], Layer-Wise Relevance Propagation [2], Shapley Regression Values [15], Shapley sampling [26], and Quantitative Input Influence [8] that have been proposed which use explanation models similar to the form in Definition 1 (for further technical details, refer to the cited papers). However, each method has its own limitations. Overall, they fail to satisfy the key interpretability/fairness properties 1-3. For instance, Shapley Regression values are computationally expensive since they require retaining the model for every subset of features (i.e., if the data contains n total number of features, then Shapley Regression values require retaining the model f 2^n times. For a large dataset, it becomes computationally expensive as the number of the features increases). In contrast, LIME is faster, but it violates the consistency property, which may cause misinterpretation of the predictions.

2.3 SHAP(SHapley Additive explanations)

To address these issues, Lundberg and LEE unify all the above methods and introduce SHAP (SHapley Additive explanations) values, which are the Shapley values of a conditional expectation function of the original model in equation (2.4) [19]. From the name itself, "SHapley" refers to the principles taken from cooperative game theory, Shapley values. In the context of machine learning, each feature is treated as a "player" in a game that contributes to the final prediction (the reward). In addition, the word "Additive" refers to the structure of the explanation model since it expresses the model output as a sum of contributions from each input feature. SHAP explains how to get from the base value (which would be predicted if we did not know any features) to the current output $f(x)$. Lastly, "exPlanations" highlights SHAP's aim to provide clear explanations for not only individual model predictions, but globally as well. Its ability to provide not only deep insights into individual model predictions, but also a global analysis of model's behavior, and to ensures that feature contributions are fairly and transparently attributed is what separates SHAP from other methods.

Lundberg and Lee formalize this framework with a small modification to equation (2.4). In the previous section, we discussed how Lundberg and Lee proposed that an explainable model g whose feature contributions (ϕ_i) are the Shapley values as in equation (2.4),

$$\phi_i(f,x) = \sum_{z' \subseteq x'} \frac{|z'|!\,(M-|z'|-1)!}{M!}\left[f_x(z') - f_x(z'\backslash i)\right],$$

is the only explanation method that satisfies the necessary properties to be considered "fair" and reliable. With SHAP values, they express feature attributions as the Shapley values of a conditional expectation function of the original model, by defining:

$$f_x(z') = f(h_x(z')) = \mathbb{E}[f(z) \mid z_S],$$

where S is the set of nonzero indices in z'. To compute $f_x(z')$, a mapping function is applied such that $h_x(z') = z_S$, where z_S contains missing values in the indices not in S. Since most machine learning models cannot handle missing values, Lundberg and Lee approximate $f(z_S)$ using the conditional expectation $\mathbb{E}[f(z) \mid z_S]$, which averages the model's output over all possible values of the missing features given the observed ones.

Lundberg and Lee acknowledge that the exact computation of SHAP values is computationally challenging, since it requires evaluating all possible subsets of input features. To address this, they propose a few approximation strategies that assist in the computation of SHAP for real-world applications, such as Kernel SHAP and Deep SHAP. Furthermore, the authors discuss that feature independence and model linearity are two optional assumptions that can be made to reduce the computational burden. For a detailed explanation of these methods, the reader is encouraged to refer to the original paper by Lundberg and Lee [19].

2.3.1 Calculation of SHAP Values

To better undertand how SHAP values are computed, we extend the previous arbitrary example. Assume that we have a data that contains the following features: Age, BMI, Gender, and suppose that the goal of the model f is to predict the probability that an individual would have a stroke based on the input features.

Similar to the computation of Shapley values, we consider each feature to be a player. For this specific individual, whose feature vector is $x = [25, 30, M]$, to compute the contribution of the feature Age on the model prediction, we need to measure the marginal contribution of Age to the model prediction. Now, we will analyze equation (2.4): We start by iterating over all possible subsets of x.

- $i =$ Age, f is the predictive (black-box) model, and $x = [25, 30, M]$.
- Note that $x' = x$ in equation (2.4) since it is a tabular data. If x was an image data, then x' would be the simplified input of super-pixels (meaningful features).
- Since we are summing over $z' \subseteq x$, it means we need to iterate over all possible subsets of x which include Age in it:
  - {Age}: this subset is represented as $z' = [1, 0, 0]$, and $z' \setminus i = [0, 0, 0]$.
  - {Age, BMI}: this subset is represented as $z' = [1, 1, 0]$, and $z' \setminus i = [0, 1, 0]$.
  - {Age, Gender}: this subset is represented as $z' = [1, 0, 1]$, and $z' \setminus i = [0, 0, 1]$.
  - {Age, BMI, Gender}: this subset is represented as $z' = [1, 1, 1]$, and $z' \setminus i = [0, 1, 1]$.

– Note: If x contained 0 in it, the nonzero entries of z' are a subset of the nonzero entries of x.

The model f in the problem takes three inputs: Age, BMI, and Gender. When one wants to compute what the model would predict if only Age and Gender are known, then this could be a problem since most models cannot handle random patterns of missing values. To avoid this, Lundberg and Lee approximate the value of $f(z')$ with $\mathbb{E}[f(z) \mid z_S]$, where $S = \{1, 3\}$. Consider,

$$z' = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \quad \longrightarrow \quad h_x(z') = z_S = \begin{bmatrix} 25 \\ 0 \\ M \end{bmatrix},$$

where $S = \{1, 3\}$ is the set of indices of the nonzero entries in z'. Hence,

$$f_x\left(\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}\right) = f\left(\begin{bmatrix} 25 \\ 0 \\ M \end{bmatrix}\right) = \mathbb{E}[f(z) \mid \text{Age=25}, \text{Gender=M}].$$

This implies that we want to predict what would be the expected probability of an individual having a stroke given that their age is 25 and they are male. Since one cannot exclude a missing variable from the model, conditional expectation is applied. The known feature values are fixed (for Age and Gender), and for the missing feature (BMI) several plausible values are sampled from the BMI of the training data. For each sampled value, a prediction is made using the model f. Once all the predictions are made, we take the average, which gives us the expected prediction given the fixed known features.

2.4 Visualizing Direct Influence

In the previous sections, we outlined the theoretical foundation of SHAP, focusing on how it interprets individual model predictions by attributing contributions to input features. To facilitate its use in practical, real-world applications, Lundberg introduced an open-source Python library called `shap` [19]. When applied to a dataset, SHAP produces a matrix of SHAP values that matches the dimensions of the input data, where each entry quantifies the contribution of a particular feature to a specific prediction. The SHAP value for each

	Feature 1	Feature 2	Feature 3	$\cdots$
Row 1 (Instance 1)	SHAP value	SHAP value	SHAP value	$\cdots$
Row 2 (Instance 2)	SHAP value	SHAP value	SHAP value	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Table 2.1: The SHAP value matrix is structured such that each row corresponds to a data instance from the dataset, with each entry representing the contribution of a specific feature to that instance's prediction.

feature can be positive or negative, indicating whether that feature increases or decreases

the predicted outcome. The sum of all SHAP values for a given instance, plus the base value, equals the model's prediction. In regression tasks, SHAP values represent how much each feature contributes to shifting the model's output from the baseline (the expected value of the model output across the dataset) to the final predicted value for a specific instance in the same unit as the prediction. While in binary classification tasks, SHAP values explain the contribution of each feature in terms of the log-odds or probability associated with the model's confidence in the positive class. In multi-class classification, SHAP produces a separate set of SHAP values for each class, showing how each feature influences the likelihood of the instance being classified into that class.

This section explores the various ways SHAP visualizes feature influence on model predictions such as beeswarm plots, waterfall plots, and bar plots plots. and explains how to interpret these figures to help make effective conclusions. Although SHAP offers a wider range of visualization options, we focus on these specific plots since they will be used in later sections of this paper. To describe the different plots, we will use an example used in one of the tutorials from the SHAP documentation [16]. It is a classification problem where the task is to predict the probability that an individual makes over $50k annually based on 12 features outlined in Table 2.2. The dataset used is the UCI adult census dataset available on Github [18] containing data on 32561 individuals. The original dataset contains 14 variables, however the dataset in shap has dropped the variable "Education" which was the categorical version of "Education-Num" making it redundant. The 'Target' column is converted to binary (True/False) where more than 50K is True (1) and less than or equal to 50K is False (0).

Feature Name	**Type & Description**
Age	Numerical; Age in years
Workclass	Categorical; Type of employment
Education-Num	Numerical; Encoded education level
Marital Status	Categorical; Marital status of the individual
Occupation	Categorical; Type of occupation
Relationship	Categorical; Relationship status
Race	Categorical; Ethnicity of the individual
Sex	Categorical; Gender of the individual
Capital Gain	Numerical; Capital gains recorded
Capital Loss	Numerical; Capital losses recorded
Hours per week	Numerical; Number of hours worked per week
Country	Categorical; Country of origin

Table 2.2: The model input variables used to predict whether a person earns over $50k per year.

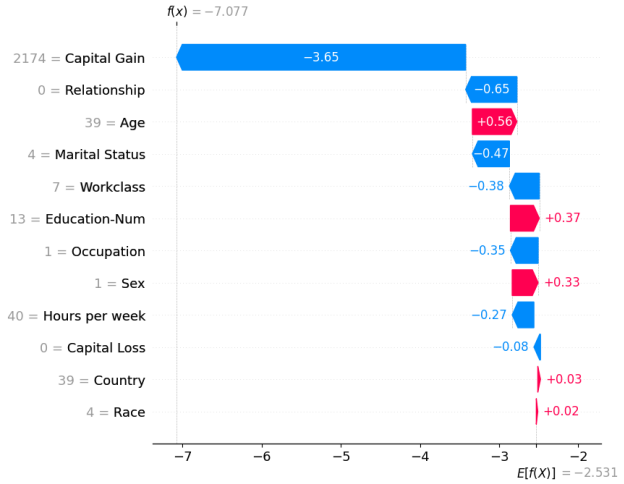


Figure 2.1: Waterfall plot for an individual case from the UCI Adult Income dataset explaining the model's prediction. On the y-axis, feature names are ordered based on importance for this specific individual, with their corresponding feature values shown in gray to the left (For instance, Capital Gain may appear at the top for this individual but not for another). The x-axis represents the SHAP values in log-odds units, as this is a classification problem. Negative values imply probabilities of less than 0.5 that the person makes over 50k annually. The plot starts from the bottom at the model's expected value $\mathbb{E}[f(x)]$, and illustrates how each feature contributes to shifting the prediction toward the final output. Positive contributions (red bars) push the prediction higher, while negative contributions (blue bars) lower it; the magnitude of each contribution is labeled on the bar.

2.4.1 Local Interpretations

Waterfall plots are very informative for making local interpretations as they will show the decomposition of how a particular prediction was made; how each feature contribution shifts the model's output from the baseline value $\mathbb{E}[f(x)]$ to the final prediction $f(x)$. This helps understand the rationale behind a specific decision made by the model. Figure 2.1 presents a SHAP waterfall plot for an individual case in the UCI Adult Income dataset, illustrating how the model arrived at its prediction. On the left side of the plot, each variable is followed by the input value, in gray, for this particular data instance. For example, this person who has a capital gain of 2174, is 39 years old, works 40 hours per week, and etc. Moreover, features are ordered from top to bottom based on their impact on the final prediction: those with the greatest contribution (positive or negative) appear at the top, while features with a smaller influence are shown toward the bottom.

Furthermore, each feature then contributes positively or negatively to shift the prediction away from this baseline; the blue bars represent negative contributions (lowering the prediction), and the red bars represent positive contributions (increasing the prediction). All the contributions, combined, add to the final model output of $f(x) = -7.077$. This value, expressed in log-odds, corresponds to a very low probability that this individual earns more than \$50k. To be exact, it is a probability of 0.0008435 that this person makes more than

50k. Hence, the model will classify "False" for this individual.

2.4.2 Global Interpretations

Bar plots and beeswarm plots are very useful for explaining the expected model behavior with respect to the whole dataset.

Bar Plots

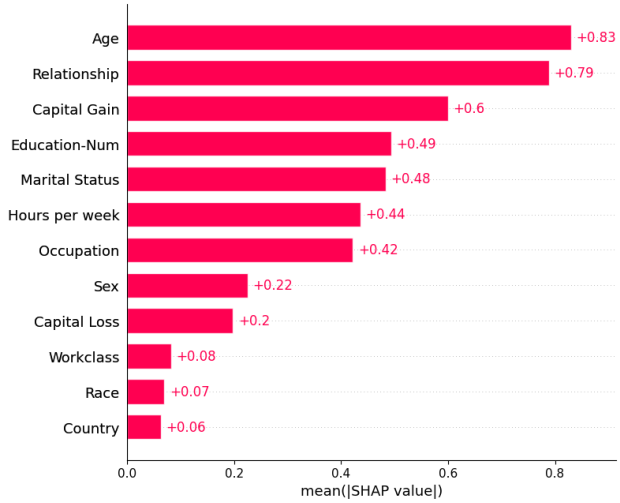


Figure 2.2: SHAP bar plot of the UCI Adult income dataset. This figure summarizes the global impact of each feature on the model's predictions. The numbers shown next to the bars correspond to the mean absolute SHAP values for each feature. The features are also ordered from most impactful to least.

Figure 2.2 represents the bar plot of the SHAP values, showing the importance of the features in order from the most influential to the least influential features. The features that appear at the top, such as Age and Relationship, on average contribute the most to the model's prediction. This helps identify which variables the model relies on most heavily when making predictions. This importance is also quantified using the mean absolute SHAP value for each feature, calculated across all given samples, which reflects the feature's average influence on the model's output (regredless if it is a negative or positive influence). For instance, we can interpret that Age contributes, on average, ± 0.83 to each predicted likelihood.

In addition to the mean absolute SHAP value, to assess feature importance, we can also generate bar plots using maximum absolute SHAP value; this highlights features that can have a great impact in individual cases, while not having significant influence across all predictions. Observe from Figure 2.3 that the Capital Gain and Capital Loss features are the top two features due to them having high magnitude effects. In contrast, in Figure 2.2, where features are ordered by their mean absolute SHAP values, Capital Loss is near the bottom which indicates that Capital Loss does not have a huge effect on the model's predictions, however for some instances, its effect can be quite strong.

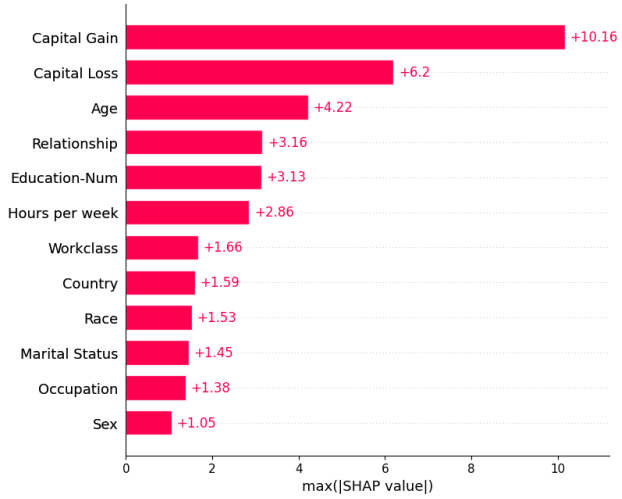


Figure 2.3: Maximum Absolute SHAP bar plot of the UCI Adult income dataset. This approach emphasizes features that have a substantial impact in specific individual cases, even if their overall influence on the model's predictions is not significant.

Beeswarm Plots

As previously discussed, bar plots allow for interpreting feature importance by illustrating their average influence on model predictions. However, it does not provide any information on the relationship between the feature influence and the underlying feature value. To gain this insight, we would want to observe a SHAP beeswarm plot as shown in Figure 2.4.

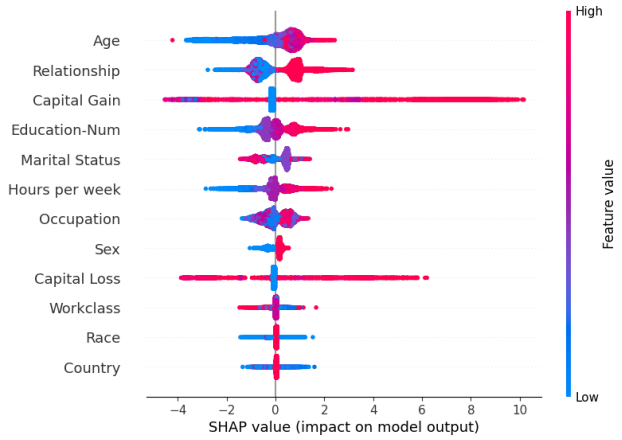


Figure 2.4: SHAP beeswarm plot of the UCI Adult income dataset. This figure demonstrates the relationship between the feature effect with the underlying feature value. Each dot is an individual sample's SHAP value (x-axis). The color signifies the feature value (red indicates high feature value, blue indicated low feature value), and the features are ordered by their mean absolute SHAP values. Meaning, Age has the greatest average impact over all the data points, and younger people are less likely to make more than $50k.

Beeswarm plots are more complex and information-rich display of SHAP values that reveal

not just the relative importance of features, but their actual relationships with the predicted outcome. In Figure 2.4, each dot represents an individual sample's SHAP value for a given feature, and the dot color represents the underlying feature value, with blue indicating lower values and pink indicating higher values. This allows for simultaneous interpretation of both the impact magnitude and its relationship to the feature value. The order of the features is, by default, according to the mean absolute value of SHAP values. So, we would know from Figure 2.4 that Age has the greatest impact over all the data points on average (which aligns with Figure 2.2), and that younger people are less likely to make more than \$50k. The length or the thickness of the shapes do not provide any information regarding the importance of the features. For instance, the spread observed for Capital Gain indicates that this feature has a high impact on specific individual cases. This aligns with the Maximum Absolute SHAP bar plot.

By also sorting the features based on the maximum absolute value of the SHAP values, this would place the features that have high impacts on individual people at the top.

2.5 Numerical Experiments

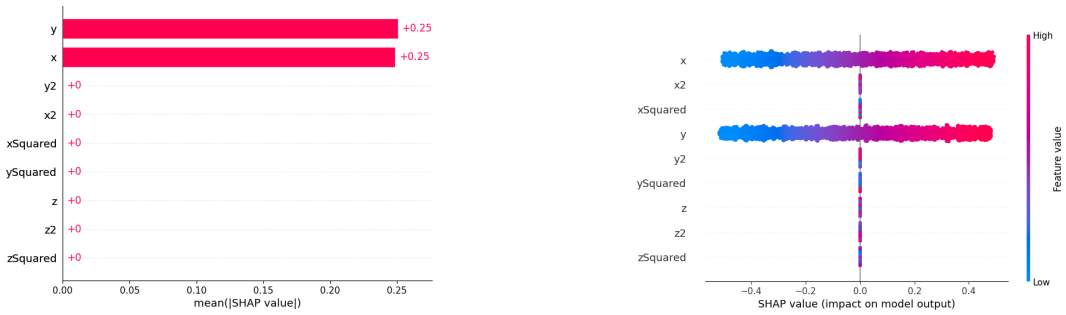
After gaining a theoretical background on measuring direct influence through SHAP values, we will now apply it on a series of experiments to develop an intuitive understanding of how SHAP values capture direct feature influence and observe how it attributes influence to input features across different settings using the SHAP Python package [17]. The goal is to gain insight into how SHAP behaves in practice, starting from simple synthetic functions and progressing to real-world datasets of increasing complexity such as the Adult Income dataset and Montréal Housing dataset. Additionally, we will also perform a convergence analysis on SHAP values, to examine the minimum number of data points needed to get accurate SHAP values. This is particularly relevant in settings where data is limited and our goal is to assess how sensitive SHAP is to data scarcity, that is, how data intensive the method is in producing reliable attributions.

2.5.1 Synthetic Numerical Experiments

To evaluate the benefits of SHAP values and use the different plots discussed in Section 2.4 to observe global and local levels of feature influence, we will begin with a simple synthetic $x + y$ experiment following the setup of Marx et al. [21], with a known model.

$x + y$ Dataset: The Model and the Data

This experiment is a regression problem, where the task is to predict $x + y$ from a synthetic dataset that contains the engineered features: x, $2x$, x^2, y, $2y$, y^2, $z, 2z$, and z^2. This dataset of 5000 samples is generated from two independent features x and y, which are drawn from a uniform distribution over the interval $[0, 1]$. The target variable defined as the sum $x + y$. The dataset also includes proxy variables $2x, x^2$, $y, 2y,$ and y^2. Additionally, an independent noise variable z, independent of x and y, is also sampled uniformly from $[0, 1]$. The inclusion



(a) (b)

Figure 2.5: Figure 2.5a presents the bar plot of the synthetic $x + y$ data, where the model has a fixed weight of 1 for x and y, and 0 for the other features. The bar plot highlights that x and y have approximately the same mean absolute SHAP value of 0.25, indicating equal contribution on average to the model's predictions. Figure 2.5b displays the corresponding beeswarm plot of SHAP values, providing more insight into the relationship between the feature values and the SHAP values. Both figures confirm that only x and y have influence on the model's output.

of these noise terms $z, 2z$, and z^2 allows us to evaluate SHAP's ability in distinguishing between important and irrelevant feature influence. Lastly, the model M used for prediction is handcrafted to directly compute $x + y$ by assigning a fixed weight of 1 to features x and y and a weight of 0 to all the other features. Formally, the model can be expressed as

$$M(\mathbf{x}) = \mathbf{x} \cdot \mathbf{w},$$

where $\mathbf{x} \in \mathbb{R}^9$ is the input data, and $\mathbf{w}$ is a column weight vector with ones in the indices corresponding to x and y, and zero everywhere else.

Results To visualize the direct influence of each feature, we produced both beeswarm and bar plots using SHAP's built-in visualization tools, as presented in Figure 2.5b and Figure 2.5a, respectively. These plots provide a global view of the feature attributions across the dataset, with the beeswarm plot displaying the distribution of SHAP values per feature and the bar plot summarizing their mean absolute contributions. Both visualizations in Figure 2.5 confirm that only x and y have influence on the model's output, as expected. Given that the model is a linear function with equal weights of 1, this indicates that these variables are equally important. The SHAP bar plot reflect this symmetric contribution in Figure 2.5a with both features having a mean absolute SHAP value of around 0.25 (the exact value for x is 0.24866773, and for y is 0.250543981), implying that on average they both have the same influence across all the data, and no direct influence for the proxy variables and the noise terms. On the other hand, the beeswarm plot shows the relationship between the feature values and the SHAP values. For instance, we can observe that higher feature values (red dots) have a higher SHAP value, i.e., more influence.

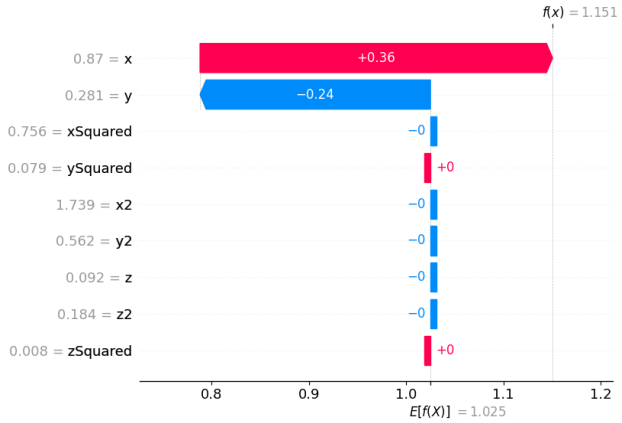


Figure 2.6: This figure presents the SHAP waterfall plot of the first instance of the synthetic $x+y$ dataset. It displays the contribution of each feature for this specific prediction, starting from the base value $\mathbb{E}(f(x))$ and showing the additive effect of each feature's SHAP value that totals to the model's final output.

By also examining the SHAP waterfall plot in Figure 2.6 of the first instance of the $x + y$ dataset, we obtain an understanding on how each feature contributed in the decision making process. This would also allow us to make references with the bar and the beeswarm plots. Starting from the base value, which represents the model's expected output in the case if it has no information about the feature values, we observe that neither the proxy variables nor the noise terms exert any influence, which is consistent with the model's design. Only the features x and y contributed to the final prediction, with x having more contribution in this particular prediction (where the values of x and y are 0.87 and 0.281 (in gray), respectively), which aligns with the patterns captured in the beeswarm plot in Figure 2.5b, where higher feature values are associated with greater SHAP values.

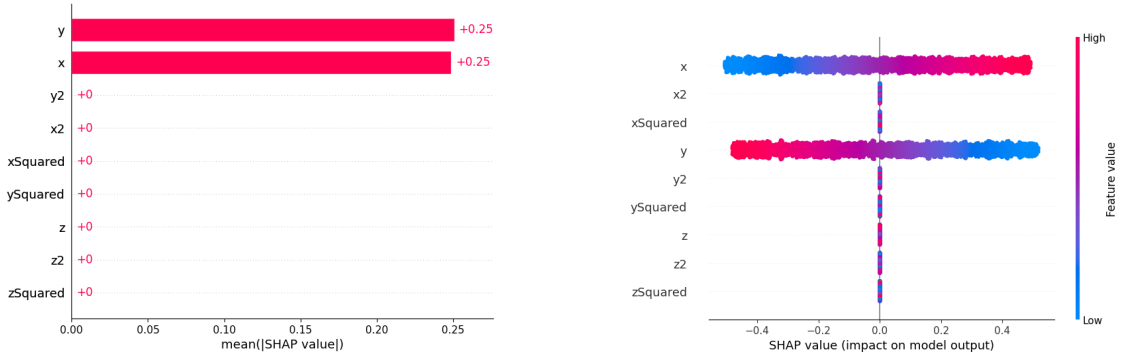
2.5.2 Different Operations on the Synthetic Dataset

Using the same synthetic dataset, we extend our analysis by applying SHAP to different algebraic target operations, such as $x - y$, $10x + y$, $-10x - 10y$, and $x \cdot y$. We will further utilize the dataset to apply SHAP on a simple linear regression model that we fit on the data. These simple yet insightful experiments allow us to understand how SHAP responds to changes in feature magnitude and direction. For all of these cases, we use the same model structure M, modifying only the weights to match the coefficients of the expressions. For the multiplicative case $x \cdot y$, a separate function was defined to compute the product of the feature values at the x and y indices.

Results Figures 2.7, 3.8, and 2.9 present SHAP bar and beeswarm plots for the different algebraic variations $x - y$, $10x + y$, $-10x - 10y$, and $x \cdot y$, respecitively, based on the same synthetic dataset as $x + y$ (Section 2.5.1).

For linear models of the form $M(x) = \sum_{i=1}^{n} w_i x_i + b$, with n features, where w_i denotes the

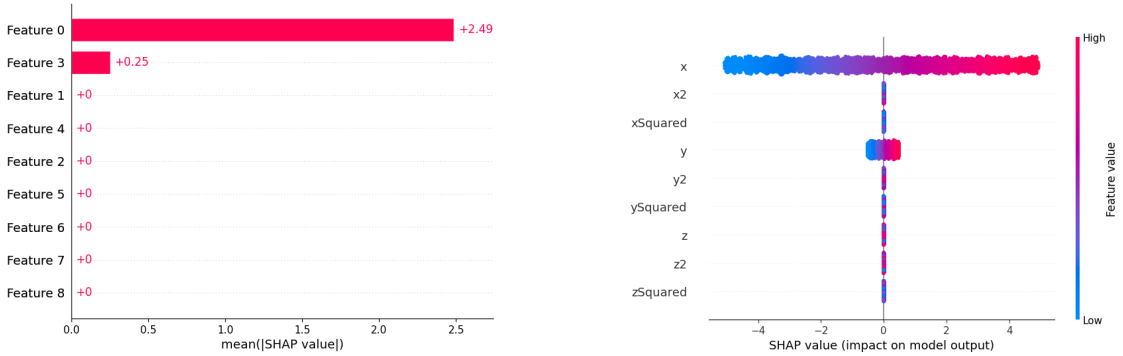
weight associated with feature x_i, we will observe that the results are consistent with Lundberg and Lee's [19] corollary which demonstrates that SHAP values are proportional to the model's weight coefficients, under the assumption of feature independence. Specifically, they show that the direct influence of a feature x_i can be expressed as $\phi_i(M, x) = w_i(x_i - \mathbb{E}[x_i])$. In our case, since x and y are drawn independently from a uniform distribution, the expected value of each is $\mathbb{E}[x_i] = \frac{0+1}{2} = 0.5$.



(a) (b)

Figure 2.7: Synthetic dataset with a target to predict $x - y$. Figure 2.7a and Figure 2.7b show the corresponding SHAP bar and beeswarm plots, respectively.

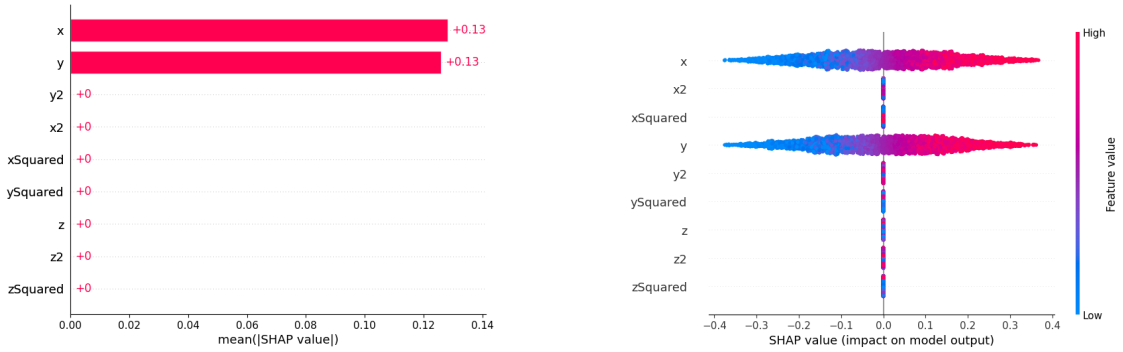
The x − y case (Figure 2.7): although the magnitude of the weights remains the same, the coefficient of y is negated. Consequently, while the bar plot in Figure 2.7a shows no change in the magnitude of the global influence (since SHAP uses mean absolute value), the beeswarm plot in Figure 2.7b clearly captures the change in the direction of the effect; higher values of y now correspond with more negative SHAP values.



(a) (b)

Figure 2.8: Synthetic dataset with a target to predict $10x + y$. Figure 2.8a and Figure 2.8b show the corresponding SHAP bar and beeswarm plots, respectively.

The 10x + y case (Figure 3.8): the weight of x is scaled by a factor of 10, greatly increasing its contribution relative to y. This is clearly evident in the SHAP bar plot in Figure 2.8a, where the mean absolute SHAP value for x is 10 times the base case $x + y$. Furthermore, the beeswarm plot reveals a wider spread of SHAP values for x, which can be observed that the SHAP values on the x-axis changed from a maximum of 0.4 to 4, showing feature scaling effect and confirming with the corollary of Lundberg and Lee.



(a) (b)

Figure 2.9: Synthetic dataset with a target to predict $x \cdot y$. Figure 2.9a and Figure 2.9b show the corresponding SHAP bar and beeswarm plots, respectively.

The final case x · y (Figure 2.9): introduces non-linearity since the model consists of the interaction term between x and y. Although both the bar and the beeswarm plots highlight that x and y are the only features directly influencing the output, one would have to be more careful in drawing conclusions. Clearly, we observe from Figure 2.9b that the distribution of the SHAP values for the features is different than the previous cases; it is denser in the middle, than at the tails, indicating that there are less data with larger feature values associated with high SHAP values. Overall, we cannot make a global implication that the data samples with larger values for x have higher value of $x \cdot y$ without considering the context; if y is approximately 0, then $x \cdot y$ will also be approximately 0. This is why it is also crucial to consider the interaction effects. To observe the relationship between the SHAP values and the interaction of the features, Figure 2.10 shows the dependence plot between x and y. We observe that for high values of x, the SHAP values depends on the value of y: when y is also large, the SHAP value of x is high, however it

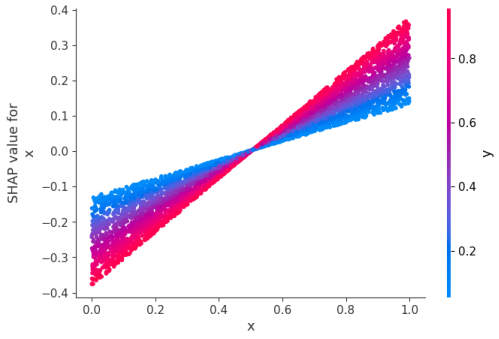


Figure 2.10: SHAP dependence plot for feature x, colored by the feature values of y (red indicates higher values of y, and blue indicates lower values). The x-axis represents the feature values of x, while the y-axis corresponds the SHAP values assigned to x. The vertical spread at each value of x reflects the model's dependency on y.

gets smaller as y gets smaller.

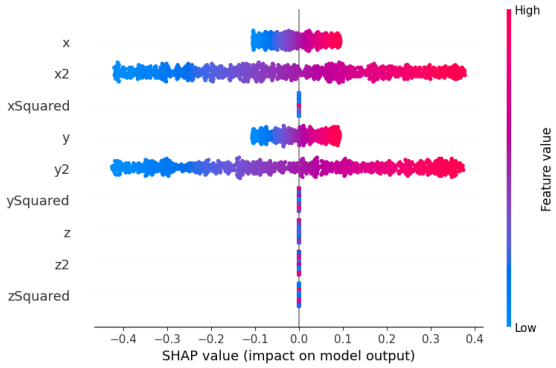
Linear Regression Model: Apart from the variations of the algebraic alterations of the base case $x + y$, we also apply SHAP on a linear regression model that we fit on the $x + y$ dataset.

Training of the model We split the data into 80% training and 20% test sets (without scaling the data), then fit a model on the training set. The weight coefficients of the features from the learned model can be obtained from the Table in Figure 2.11a. We obtained a Mean Square Error on the test set that is negligible within machine precision, indicating an extremely good fit. However, fitting the model is not the main priority; rather, the objective is to observe the insights that we will obtain from SHAP).

Results Applying SHAP on the learned model, we obtain the visualization in Figure 2.11b, which shows direct influences for not only x and y, but the proxy variables $2x$ and $2y$. This is expected, as the fitted model is a linear model, and these models are sensitive to the linear patterns in the data; which is why we see activity for $2x$ and $2y$, but not for x^2 and y^2. Now, since we are aware that the model is predicting $x + y$, the beeswarm plot in Figure 2.11b highlights the redundancy in feature representations.

Feature	Coefficient
x	0.19999999999999998
x2	0.4
xSquared	2.498001805406602e-16
y	0.19999999999999984
y2	0.3999999999999999
ySquared	-4.1492417141020255e-16
z	-4.206704429243757e-17
z2	-6.114900252818245e-17
zSquared	2.8601253310167607e-16

(a)



(b)

Figure 2.11: Figure 2.11a presents the coefficients of the linear regression model that we fit on the $x + y$ dataset. Figure 2.11b displays the corresponding beeswarm plot.

These visualization and comparisons of the different modifications applied on the base $x + y$ dataset demonstrate SHAP's ability to capture the magnitude and direction of feature contributions. Not only this, but SHAP is also valuable for debugging and serves as a sanity check; by capturing unexpected feature influences, it can prompt deeper analysis of the data quality, feature engineering, or model behavior. Especially in more complex or real-world situations, where it is more common to deal with black-box models whose structure is unknown, SHAP can assist in identifying which features the model is directly leveraging to make its predictions.

2.5.3 Adult Income

Similar to Marx et al, we also examine a real-world dataset known as the Adult Income dataset which is collected from the 1994 U.S. Census [3]. This dataset is widely used in the fairness-aware machine learning literature, and contains 14 features on employment type, demographic characteristics, and financial indicators such as capital gains and loss. The classification tasks is to predict whether an individual makes an annual income of more than $50k.

The dataset already comes separated into a train and test sets. However, during the data cleaning and training step, a few issues were encountered with the predefined test set. Hence, we decided to use only the training set for this experiment. To preprocess the data, we followed the steps of Marx et al. Complete details are provided in the Appendix . Next, we trained a Logistic Regression to predict whether individuals make an annual income of more than $50k, and obtained the following losses on the test sets: BCE loss = 0.3265, and test accuracy of 84.97%, which are the same losses as the ones of Marx et al, implying that the choice of dataset did not affect the results.

Figure 2.12 presents the SHAP beeswarm plot generated to analyze the direct influence of features on the predicted probability of an individual earning more than $50k per year. For categorical features, which are one-hot encoded, low values (represented by the blue points) indicate that the corresponding feature is False. From the Figure, we observe that for capital gain, the points are more spread out. This indicates that there are some individuals with very high capital gain and for these individuals, the probability that they make more than $50k is higher. Similarly, for the variable "age" and "hours per week" on average seem to have a positive impact across the data since we can observe that red dots correspond with the positive SHAP values, indicating that for these variables, the probability of making more than $50k increases for an older. The Figure also shows that the feature representing the number of hours worked on average impacts positively. Interestingly, there are some features that appear to have little to no direct influence on the model (or they seem to be not influencing the model on the surface). For instance, the feature race (with its different categories) seem to have minimal impact on the predicted outcomes. It is crucial to be mindful of such results when analyzing due to the historic pattern of discrimination that surrounds race and gender with income. To do a more in-depth analysis, one could also generate dependence plots to observe the relationship between the features. Lastly, it would be beneficial to observe the indirect influence of features (which is conducted in Chapter 3) as some features although seem to appear non-influential, but might be indirectly impacting the outputs. It should also be noted that the difference between the features and the color of the SHAP values in the beeswarm plot would be a sign if the corresponding feature is continuous or a categorical binary feature. The binary features are represented with only red and blue (no smooth transition), while the we do not see similar separation for continuous variables.

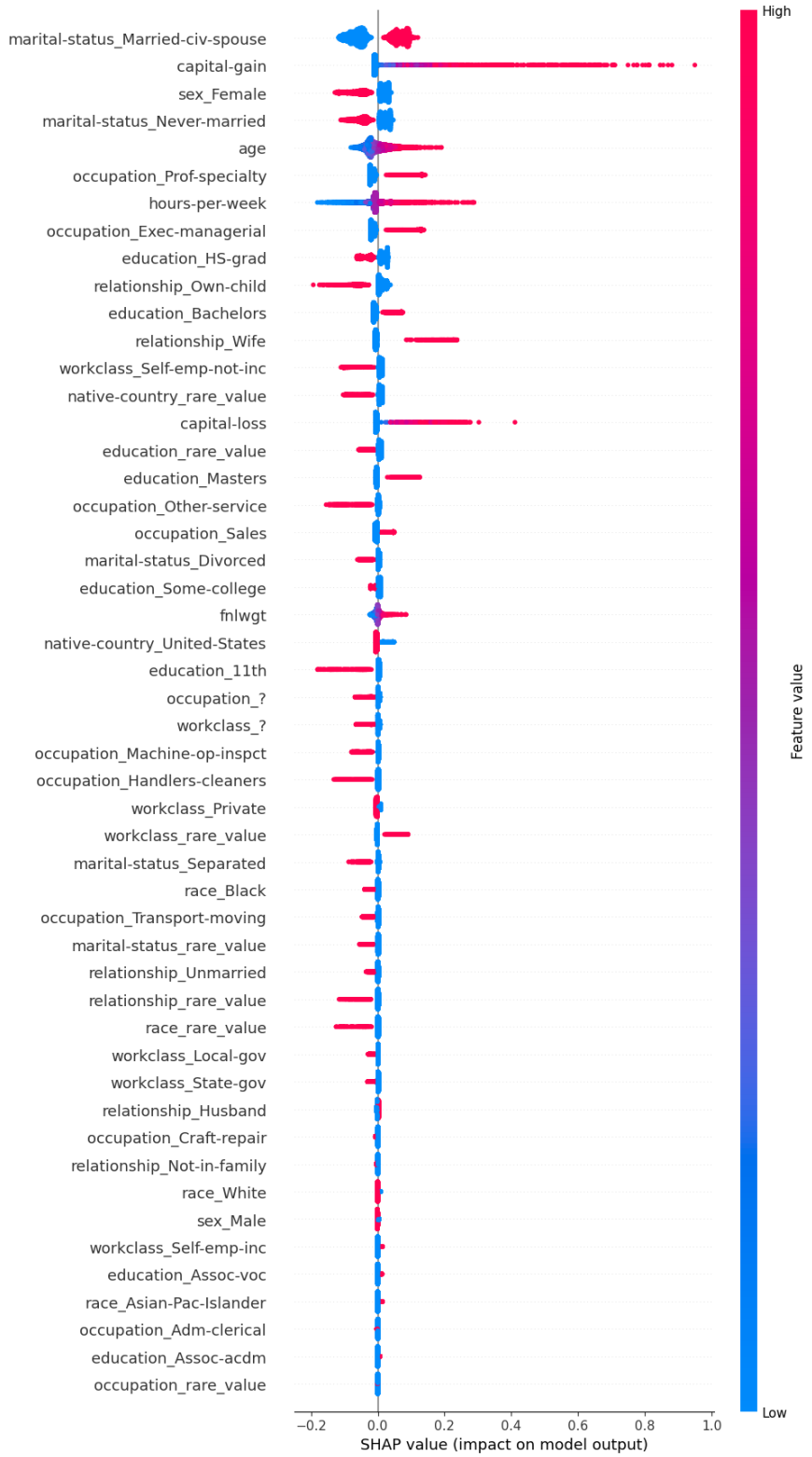


Figure 2.12: SHAP beeswarm plot to visualize the direct influence of the features to the predicted probability of the positive class (the probability that an individual makes an annual income of more than \$50k) in the Adult Income Dataset. Higher SHAP values indicate that the corresponding feature would increase the probability of the person being classied as having more than \$50k per year.

2.5.4 Montréal Housing Dataset

The housing market is no different when it comes to the use of ML models for decision making processes. As a result, we wanted to apply the acquired techniques to analyze a real-world dataset of properties from the Montréal housing market. The dataset includes detailed information for each property listing obtained from Centris [6]. There are 23 features describing the type of building, land area (superficie du terrain), total number of rooms (pièces), and region (quartier), property price (asking price), as well as Local Logic scores (which are scores on a scale of $0-5$ that rate different aspects of a location such as proximity to schools, public transit, groceries, parks, nightlife, and more). Details on the features can be found in the Appendix. Prior to model training, standard preprocessing steps were performed: categorical variables including the building type and quartier were one-hot encoded, numerical features were standardized, the target variable price was log-transformed, and missing values and extreme outliers were removed. Also, it is important to note that the focus of this experiment was not on optimizing model accuracy but on analyzing feature influence effects, which is discussed in the next chapter.

In regression tasks, SHAP values explain how much each feature increases or decreases the predicted price relative to the model's average prediction. Whereas, for classification tasks, SHAP values explain how much each feature increases or decreases the probability of belonging to a specific class. To better understand how these features influence property prices, we approached this problem as a regression task, predicting the sale price based on the available input features. During the training process, log-price was used as the target variable, and stratified splitting of the data was implemented. Initially, we trained a Random Forest Regressor, obtaining a Mean Absolute Error (MAE) of \$174,036. Other models were also tested but similarly performed poorly. To achieve better results, we introduced a new feature, price per square foot, which greatly improved model performance. Table 2.3 summarizes the different performance metrics for the different models tested, where we can observe that the MAE of the Random Forest Regressor decreased to \$36,202.14.

Model	**MAE (\$)**	**MAPE (%)**	**RMSE (\$)**	$\mathbf{R^2}$
Random Forest Regressor	\$36,202.14	3.02%	\$106,971.89	0.97
Gradient Boosting Regressor (GBR)	\$38,462.57	3.90%	\$77,267.48	0.98
XGBoost Regressor	\$46,992.93	4.41%	\$105,934.38	0.97
Linear Regressor	\$185,995.17	21.10%	\$319,510.20	0.74

Table 2.3: Summary of model performance metrics of the different models tested on the Montréal Housing dataset.

Once trained, SHAP was applied to the Random Forest Regressor to analyze each feature's impact on the model behavior in predicting property prices across the dataset. The results of the SHAP beeswarm plot are presented in Figure 2.13, which indicate that the top three influential features are the superfice du terrain (land area), price per square feet, and the pieces beta (total number of rooms). Specifically, the figure shows that houses with bigger superfice du terrain tend to have higher predicted price. The remaining features did not exhibit any influence. This suggests that the model primarily uses these three features when

making its decisions, while factors such as location, property type, or Local Logic scores do not have any effect on the sale price.

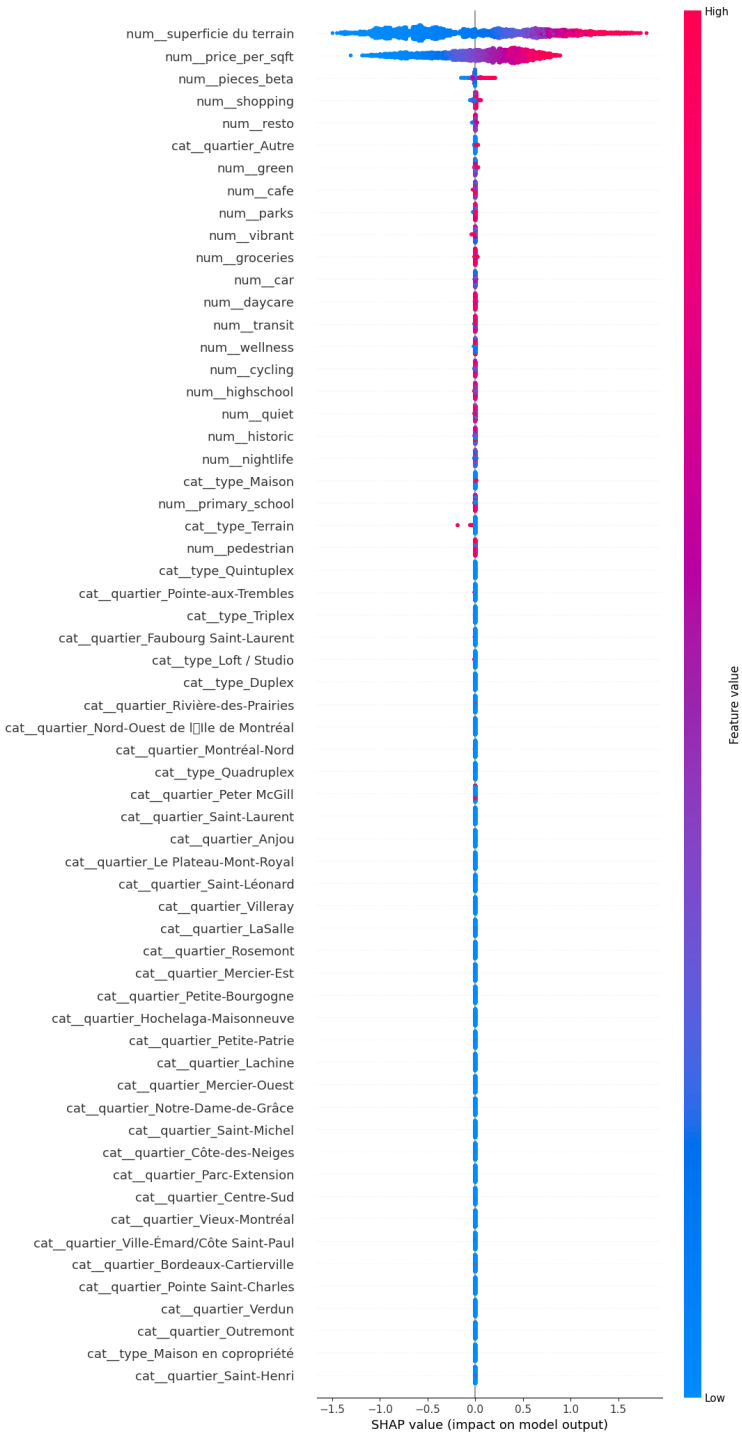


Figure 2.13: SHAP beeswarm plot of the Montreal Housing Dataset using a Random Forest Regressor to predict property prices. The features sorted in order of average importance. Feature names are prefixed with "num" for numerical features and "cat" for categorical features. The results show that the most influential features are the superfice du terrain (land area), price per square feet, and pieces beta (total number of rooms), while the remaining features exhibit no influence on model predictions. Features such as shopping, green, resto, quiet, highschool, and others are Local Logic scores, which reflect neighborhood characteristics. Higher scores indicate that the property is close to shopping centers, schools, restaurants, or it is located in a quieter neighborhood.

2.5.5 Convergence Analysis

Apart from conducting various experiments to observe the behavior of SHAP values, it is clear that SHAP shows strong performance in analyzing direct feature influence and can also be a useful tool in model debugging. SHAP values not only provide insight into the relation between feature values and model predictions, they might also expose questionable feature effects, which would prompt one to further investigate the data or the model.

In this thesis, one question that we were also interested in exploring is how data hungry are SHAP values. To investigate this, we begin this section with an error analysis, and then follow with convergence analysis where we examine convergence of SHAP values as a function of the number of samples required.

We bring back the synthetic $x + y$ dataset, which contains 5000 samples, to analyze the convergence of the SHAP value estimates. In this controlled setting, by applying SHAP to explain the model predictions, we obtain the matrix of SHAP values for each feature, as shown in Table 2.1. For a given feature i, let s_i be its true SHAP value. For every feature (column), the table computes the SHAP value for every data instance (row). In this case, we consider the true s_i to be the mean of the absolute SHAP values across the 5000 samples. Let $\hat{s}_i$ be the approximation of the true SHAP value s_i.

In order to understand how many samples n we need in order for the SHAP explainer to produce accurate results in estimating the true SHAP value, we perform the following: We define a set of sample sizes of the form $n = 2^k$ for $k = 1, \cdots, 11$. For each sample size (n), we randomly sample from the dataset and average the absolute SHAP values of the corresponding sample. Then, we repeat this process 20 times. At the end of the process, we get left with 9 matrices of size 20×11, where the columns are the sample sizes, and for each sample size we computed 20 mean absolute SHAP value estimates.

The distributions of the estimated mean absolute SHAP value for features x and y are presented in Figure 2.14. The top row illustrates the distribution of the approximations for all the sample sizes, while the bottom row offers a closer view of the larger sample sizes. The red dotted lines represent the true mean of the absolute SHAP values of their corresponding features, where $s \approx 0.24866$ for x, and $s \approx 0.2505$ for y. As expected, by looking at the figures in Figure 2.14, we observe that using smaller sample sizes (in this case, less than 2^8) will not produce accurate estimates of the true s_i's. We continue further by also observing the distribution of the absolute error of the estimates, defined as $|\hat{s}_i - s_i|$, for both features. These estimation errors are visualized through the generated boxplots in Figure 2.15, where the y-axis is plotted on a logarithmic scale to better capture the range of error magnitudes. This figure aligns with Figure 2.14 in confirming that estimations improve for bigger sample sizes as shown by the decrease in the errors.

Although Figures 2.14 and 2.15 illustrate that SHAP value estimation improves by increasing the sample size, we now turn to a more numerical analysis perspective. Specifically, we investigate the order of convergence of the mean absolute SHAP value estimates with

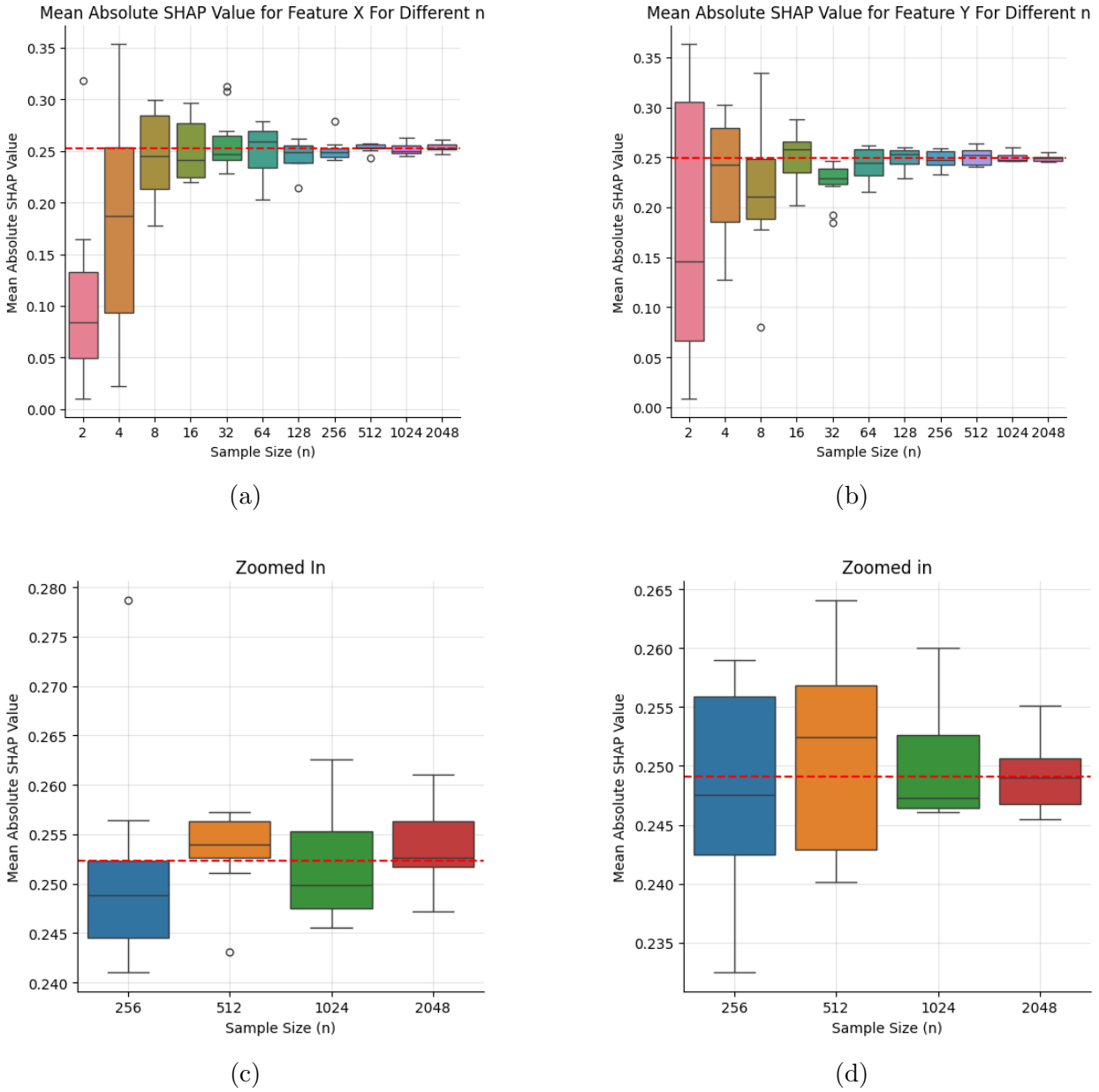
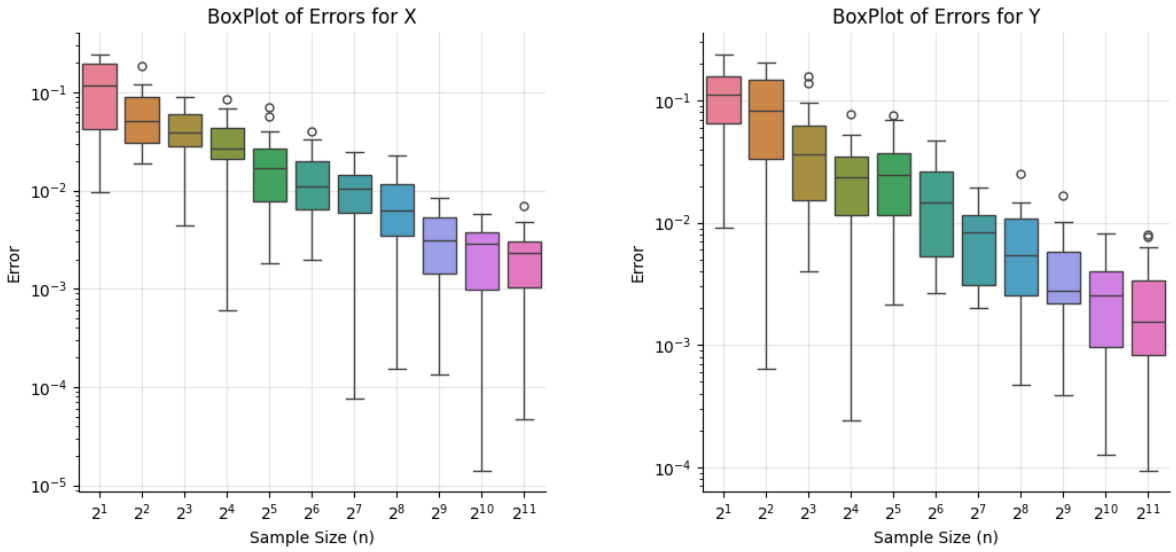


Figure 2.14: This figure presents the distribution of the estimated true SHAP value, where the true SHAP value s is the mean absolute SHAP value of the features x (left two figures) and y (right two figures). Bottom two figures provide a zoomed in version of the distributions for larger sample sizes. The red dotted lines are the true mean of absolute SHAP values, where $s \approx 0.24866$ for x, and $s \approx 0.2505$ for y.

respect to the sample size. The goal is to understand how fast the error is decreasing as a function of n, this rate is defined formally by the order of the convergence.
Recall from [5, Section 1.3] that given a sequence $\{\alpha_n\}_{n=0}^{\infty}$ that converges to some number α, and $\{\beta_n\}$ is a known sequence that converges to 0. If there is a positive number K such



(a) (b)

Figure 2.15: Boxplot of the absolute error $|\hat{s}_i - s|$ of the estimated SHAP values for features x(left) and y(right), across different sample sizes, and the y-axis is plotted on a logarithmic scale. Each boxplot contains 20 random estimation errors, and the y-axis is log-scaled.

that

$$|\alpha_n - a| \leq K \cdot |\beta_n|, \quad \text{for large } n,$$

then we say that the sequence α_n converges to α at an order of convergence $O(\beta_n)$. Commonly, $\beta_n = \frac{1}{n^p}$, for some $p > 0$.

By generating a log-log plot of the the absolute error against different powers of the sequence $K \cdot \beta_n$, one can infer the order of the convergence of the absolute errors by comparing slopes of the resulting lines. In the log-log plot, the lines would be linear, and parallel lines have the same slopes [4]. Therefore, the two parallel lines would have the same order of the convergence.

Suppose we have the absolute error as a function of n, where

$$\epsilon(n) = K \cdot \beta_n, \quad \text{and } \beta_n = \frac{1}{n^p}.$$

To plot this on a log-log plot, one would have to apply log transformation on both sides. This gives the following,

$$\log(\epsilon(n)) = p \cdot \log(\frac{1}{n}) + \log(K),$$

where p is the slope of the line (i.e., the order of the convergence and the bigger the value of p, faster the convergence rate), and $\log(K)$ is some constant that does not affect the slope of the line. Furthermore, Figure 2.16 illustrates the convergence analysis of the absolute error $|\hat{s}_n - s|$ and the sequences in a log-log plot. From this figure, we observe that the error

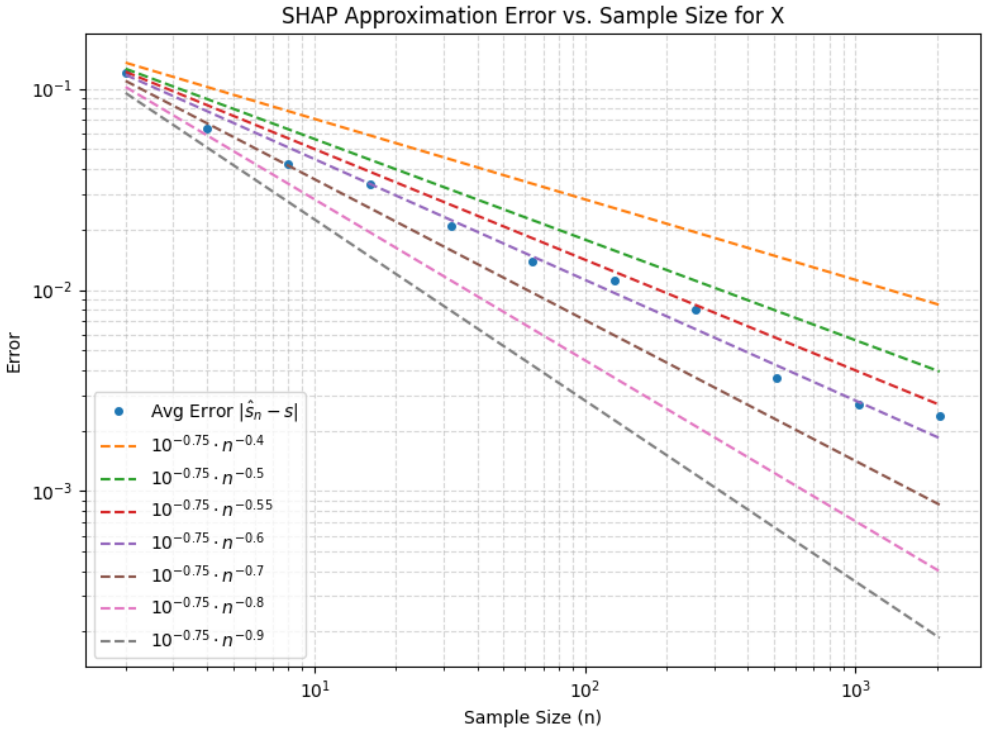
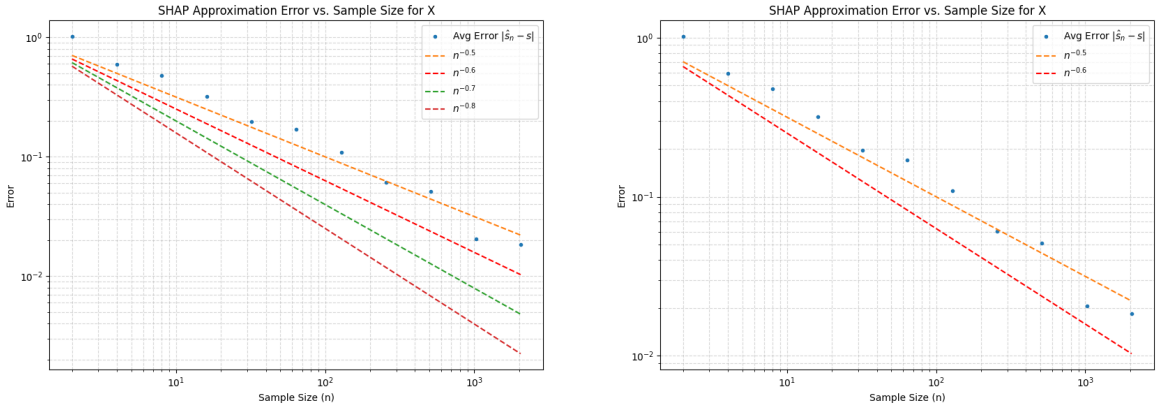


Figure 2.16: Log-log plot of the average absolute errors $|\hat{s}_i - s|$ and the sequences $K \cdot \beta_n$, where $\beta_n = \frac{1}{n^p}$ and $p > 0$. This visualization of the convergence analysis can be used to guess the order of convergence of the average absolute errors. Order of convergence of the average absolute errors is $O(\frac{1}{n^{0.5}})$.

$|\hat{s}_n - s|$ and the sequences $\frac{1}{\sqrt{n}}$ are parallel. Note that the constant $10^{-0.75}$ (which is K here) has no effect on the order of convergence. It is used for shifting the lines down. Hence, the order of convergence of the average absolute errors is $O(\frac{1}{\sqrt{n}})$. This means that to reduce the error by a factor of 10, we need to increase the sample size by a factor of 100.

We also observed the order of the convergence of the average absolute errors $|\hat{s}_i - s|$ for the $10x + y$ model. Figure 2.17 illustrates the results obtained. The figure on the left shows the comparison with multiple lines, while the figure on the right is isolated to better identify which one it is parallel to.



(a) (b)

Figure 2.17: Log-log plot of the average absolute errors $|\hat{s}_i - s|$ and the sequences $K \cdot \beta_n$, where $\beta_n = \frac{1}{n^p}$ and $p > 0$. This visualization of the convergence analysis can be used to guess the order of convergence of the average absolute errors. Left compares it with multiple sequences, whereas the figure shows the isolated version for a better comparison.

Chapter 3

Measuring indirect Influence

As machine learning becomes increasingly central to critical decision-making tasks from credit scoring to medical diagnosis, it is crucial to understand how and why these models arrive at their predictions. Numerous feature attribution techniques such as SHAP and LIME (discussed in Chapter 2), have been developed to quantify how individual input features directly impact model predictions. These methods assume a relatively straightforward relationship between features and predictions. However, this perspective is incomplete if the aim is to understand how and why ML models make their predictions.

In many real-world datasets, features exhibit complex interdependencies. A feature might influence a model's output not directly, but indirectly, through its effect on other variables, commonly referred to as proxy variables. Indirect influence has serious implications, especially when the indirectly influential feature is a sensitive attribute such as gender, race, or socioeconomic status. For instance, a model may not use race explicitly, but may still be biased if it relies on Postal codes or ZIP codes, since in segregated environments these are often dependent with race. Even if race is removed from the dataset, its influence can still remain through proxy variables (such as ZIP codes), making the model biased.

Methods for measuring the direct influence of features often fail to detect the indirect influence. Hence, in this chapter before we begin with a review of a few approaches for measuring indirect influence, specifically Black-Box Auditing (BBA) [1], and introducing the primary method explored in this thesis [21], we will provide a background overview on Adversarial Autoencoders (AAEs) in Section 3.1, which is the technique used in [21] to learn disentangled representations. Lastly, in Section 3.4, we provide the results of the experiments we conducted to audit indirect influence.

3.1 Background on Adversarial Autoencoders and Their Training Process

Inspired by the Generative Adversarial Networks [9], Makhzani et al. [20] introduced AAEs, which are a class of generative models that combine the classical autoencoder architecture with adversarial training. To understand the main method implemented in this thesis to

identify indirect influence of features, it is essential to explore the technique used, Adversarial Autoencoders (AAEs). Hence, the following sections provide an overview of AAEs to build a background foundation on their components and mechanisms.

3.1.1 What Are Autoencoders?

To understand Adversarial Autoencoders, it is important to understand what Autoencoders are. Autoencoders are a type of unsupervised neural networks which are composed of two parts: an encoder and a decoder. The goal of an autoencoder is to reconstruct the original input as best as possible (additional details about neural networks are provided in the Appendix).

Let f denote the encoder and g the decoder. Following the arrows in Figure 3.1, given a d-dimensional, data input $x \in \mathbb{R}^d$, x first passes through the encoder f, where it gets compressed into a lower-dimensional latent representation $x' = f(x)$, where $x' \in \mathbb{R}^k$. Then, the decoder g attempts to reconstruct the latent vector back to the original input, producing $\hat{x} = g(x')$, where $\hat{x}$ is the reconstruction of x. Since the goal of the autoencoder is to reconstruct x as best as possible, it is trained to minimize the reconstruction error between the input x and its reconstruction $\hat{x}$, $\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\|^2$, with the aim of reducing dimensionality while preserving the essential structure of the data. Autoencoders can learn nonlinear embeddings and are often used for tasks such as denoising [27], and recommendation systems [28].

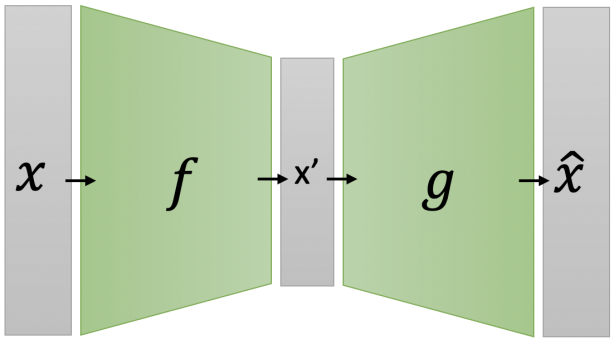


Figure 3.1: Architecture of an Autoencoder. It is composed of two parts: an encoder f and a decoder g. Given a $d-$dimensional input x, x passes through f, where the encoder applies dimensionality reduction while keeping the inherent structure of the data. The lower dimensional latent representation is denoted by x'. Lastly, x' passes through the decoder g for reconstruction.

Building on the foundational autoencoder framework, extensions such as Variational Autoencoders (VAEs) [11] and Adversarial Autoencoders (AAEs) [20] have emerged, which are extremely useful in applications such as image generation, speech and audio synthesis, Disentangled Representations, and Anomaly detection.

3.1.2 Adversarial Autoencoders

Now that we have established a foundation on what autoencoders are, we can begin by understanding its connection with AAEs. As discussed earlier, the main objective of an autoencoder is to learn a compressed representation of the input data, commonly referred to as the latent code, such that this representation retains important information to accurately reconstruct it back to the original input.

However, unlike the traditional autoencoder which only focuses on minimizing the reconstruction error without any restrictions on the distribution of the latent space (could make it hard to interpret the latent space and utilize it for tasks beyond reconstruction), AAEs aim to match the encoded latent space to a predefined prior distribution. Moreover, AAEs contain three main components: an encoder, a decoder, and a discriminator.

To refine the latent space, AAEs imposes a prior distribution $p(x')$ (where x' is the latent representation of the input), and require that the encoder's output distribution $q(x')$ approximately matches the prior $p(x')$. This is where the discriminator comes in. This alignment of the distributions is achieved through the adversarial training; the discriminator works against the encoder to ensure that the latent space distribution matches the imposed prior distribution. The goal of the encoder is to produce latent representations that are indistinguishable from the real data.

The formal definition of AAEs is this. Now, we will discuss how Marx et al. [21] utilize this framework to create disentangled representations.

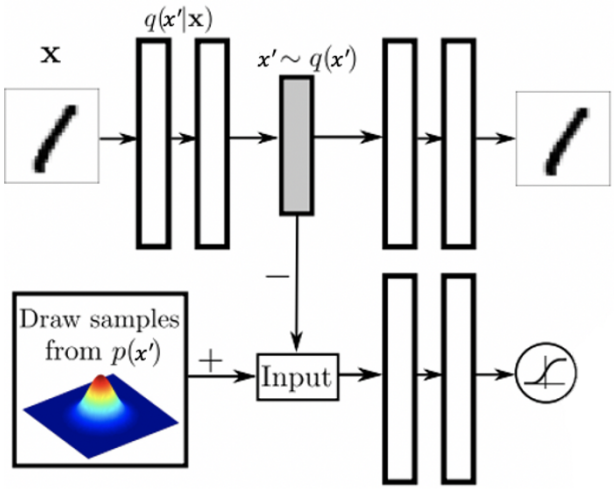


Figure 3.2: Architecture of an Adversarial Autoencoder. Figure adapted from [20].

3.2 Brief Literature Review for Indirect Influence

In the auditing literature, to quantify the indirect influence of a certain feature, a common approach involves modifying that feature (usually through random perturbation). Then, quantifying the indirect influence by measuring the difference between the accuracy of the model with the original dataset and the accuracy of the model with the perturbed dataset. However, randomly perturbing features can lead to loss of meaningful information contained in the dataset. This can negatively impact the model's performance, causing the audits to be less reliable and less representative of the feature's actual influence.

Adler et al. follow a similar approach. Although they follow the same steps, rather than incorporating random perturbations, they propose a method called Gradient Feature Auditing (GFA), which removes both direct and indirect influence of a feature by obscuring it. Obscuring a dataset involves transforming the data such that the feature of interest can no longer be predicted from the remaining features, while preserving the overall structure of the dataset. For each feature, they apply this obscuring procedure and then measure the change in model accuracy. The difference in accuracy before and after obscuring serves as an estimate of the feature's indirect influence.

Furthermore, the obscuring process is data-type dependant; the method applied to obscure the data varies depending on whether if it is a numerical or a categorical feature. In addition, Adler et al. also specify that their approach is limited to only numerical and categorical data, meaning that their procedure does not extend to image data. Lastly, Adler et al.'s proposed method takes a global perspective as it evaluates feature influence by measuring the drop in the accuracy of the prediction across the entire dataset after systematically obscuring a feature. It does not provide any explanations for individual data instances. The following Table 3.1 provides a comparison between two methods that measure indirect influence of features: BBA (Black Box Auditing) and DIA (Disentangling Influence Audits) which will be covered in Section 3.3.

3.3 Disentangling Feature Audit: A Theoretical Framework

Marx et al. [21] proposed another method, which is also the primary approach employed in this thesis, for auditing black-box models and indirect influence of features, both at the global and individual levels. In their paper entitled *Disentangling Influence: Using Disentangled Representations to Audit Model Predictions*, they introduce their modular technique that reduces the computation of indirect influence into measuring direct influence in a transformed space, where the feature of interest is disentangled from the other features. They also leverage SHAP values (as defined in Section 2.3) as a measure of direct influence. It is also important to emphasize that this framework can audit not only categorical features but also continuous features and high-dimensional inputs such as image data which either required additional preprocessing beforhand or were inapplicable with earlier approaches.

	BBA [1]	**DIA** [21]
Input data	Categorical and numerical data	Categorical, numerical, and image data
Main idea	They develop a technique for auditing black-box models, which lets them study the extent to which existing models take advantage of particular features in the dataset, without knowing how the models work	They develop disentangled influence audits, a procedure to audit the indirect influence of features, to perform both global and individual-level indirect influence audits
Global/Local perspectives	Global	Global and Local
Quantifying Indirect Influence	Measure the change in accuracy of the model with the original dataset and the perturbed dataset	The indirect influence of a feature on the prediction of a model is the direct influence influence of a feature on the prediction of the disentangled model
Method	Gradient Feature Auditing (GFA) (works feature by feature)	Disentangled Representations (DR)
Process	• Apply an obscuring procedure to each feature • Calculate the overall indirect influence of the feature	• Create disentangled representations using adversarial autoencoders • Audit direct features
Requirement of the black box model to be retained	No	No

Table 3.1: Comparison between BBA (Black Box Auditing) and DIA (Disentangled Influence Audits).

Marx et al. break down the procedure into two key steps:

- First: they construct disentangled representations of the data that isolate the feature of interest from the other variables using adversarial autoencoders.
- Second: they compute the direct influence of the disentangled representations using SHAP.

Notation

Before proceeding, we define the notations used throughout this section. Unless otherwise stated, the notation follows that of Marx et al. [21].

Given a dataset, let P denote the set of features of interest with associated domain $\mathcal{P}$. Let X denote the remaining features of the data (that may or may not be influenced by the features in P) with domain $\mathcal{X}$. A data instance is represented as a point $(p, x) \in \mathcal{P} \times \mathcal{X}$, where $p \in \mathcal{P}$ is the value of the feature of interest and $x \in \mathcal{X}$ is a vector containing the values of the remaining attributes. Furthermore, let Y denote the space of target labels. For binary classification tasks, assume $Y = \{+1, -1\}$ (though the Marx et al's framework generalizes to multiclass), or $\mathbb{R}$ for regression. Lastly, for simplicity, Marx et al. state that they will consider P to represent one feature, however, they mention that it is possible to consider more.

3.3.1 Disentangled Representations

In the process of measuring the indirect influence of features, the first step is to create disentangled representations. As we all know, in real-world datasets, there could be inter-dependencies among the features (i.e. some variables may implicitly contain overlapping information about one another). This could make it difficult to isolate its influence on the model's prediction. The objective of creating disentangled representations is to, from the name, disentangle the (entangled) feature of interest from the rest of the features, allowing for a better analysis of its indirect influence.

Disentangled representations aim to learn independent factors of variation that reflect the natural symmetries of the data [21]. Let $x' \in \mathcal{X}'$ represent the transformed x that is independent of p. Creating a disentangled representation of (p, x) means finding an invertible mapping f that takes the original input and maps it onto a feature space where the sensitive feature p and the rest x are independent. Formally, this is defined by:

$$f(p, x) = (p, x'), \quad \text{where } (p, x') \in \mathcal{P} \times \mathcal{X}'. \tag{3.1}$$

3.3.2 Construction of the Disentangled Representations

Now that we have defined what disentagled representations are, we can proceed to explain the procedure on how to construct them. As mentioned previously, Marx et al. create disentangled representations via adversarial autoencoders. Recall that AAEs are composed of three components: an encoder f, a decoder g, and a discriminator h. Each plays an important role in ensuring the independence of the transformed variables with the feature of interest p. We will now provide a more detailed explanation of the procedure and the roles of each component:

Architecture Overview

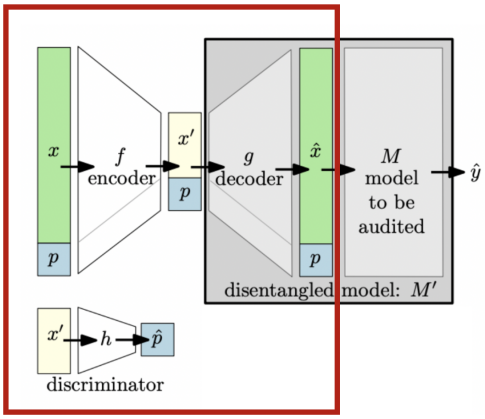


Figure 3.3: The architecture annotated in the red box of this figure displays the architecture of the autoencoder f, and g, and the discriminator h. The encoder takes the input data and produces a latent representation $x' = f(p, x)$. The decoder reconstructs $\hat{x} = g(x', p)$, while the discriminator predicts $\hat{p} = h(x')$. The encoder is trained to minimize reconstruction error while discouraging x' from retaining information about p, which is enforced through adversarial training with h. Note that p is propagated through the encoder and the decoder. However, the discriminator is not given access to p. Figure courtesy of Marx et al. [21].

- The **Encoder** takes the input (p, x), and compresses it into a lower-dimensional vector (p, x'), where x' is the lower-dimensional representation (the latent code) of the input vector x. The encoder will produce x' such that:
  - The latent vector x' preserves the inherent structure of the data so that the decoder will be able to reconstruct the original input.
  - The latent vector x' is independent of p because the goal of the AAE is to create disentangled representations. Ensuring this independence is achieved by hiding p from the discriminator (i.e., the encoder does not want the discriminator to be able to predict p which we will see in more detail in the **Training Process** paragraph below).
- The **Decoder** reconstructs the original input from (p, x'). It is to note that although the latent representation is x', (p, x') will be fed into the decoder for reconstruction, and the reconstruction is denoted as $(p, \hat{x})$.
- The **Discriminator** strives to predict p. Without having access to p, the discriminator will take in x' and outputs $\hat{p}$ with the aim of predicting p. This adversarial game between the encoder (which is trying to hide (p) and the discriminator (trying to find (p) is what enforces the independence between x' and p.

Training Process The training process of an adversarial autoencoder consists of optimizing two competing networks: the autoencoder (which is composed of the encoder f and the decoder g), and the discriminator h. This procedure is completed in two phases:

1. **Phase 1:** To make sure that the autoencoder reconstructs the original input with minimal loss (i.e., ensuring that x' retains meaningful information for the decoder for reconstruction), f and g are trained to minimize the reconstruction loss defined by:

$$\mathcal{L}_{\text{recon}} = \text{MSE}(x, \hat{x}) = \|x - (g \circ f)(x)\|^2, \tag{3.2}$$

note that we freeze the discriminator in this training steps since we want only the encoder and decoder to get updated (i.e., to better reconstruct the data).

   - Note that in this phase, the encoder also wants to hide the protected feature from the discriminator, it does so by maximizing the error of the discriminator. This is captured in the total loss function:

$$\mathcal{L}_{\text{total}} = \text{MSE}(x, \hat{x}) - \beta \ \text{MSE}(p, \hat{p}), \tag{3.3}$$

     where β is a hyperparameter controlling the strength of the disentanglement constraint. We will consider $\beta = 0.5$ as Marx et al. [21]. The reason why there is a negative in front of the discriminator error, is because although the encoder is trying to minimize the reconstruction loss (first part of the expression), at the same time it wants to produce x' that is independent of p so that the discriminator is not able to predict the protected feature from x'. The encoder does this by maximizing the error of the discriminator, which is equivalent to minimizing the negative of the discriminator error.

2. The second phase of the training process is optimizing the discriminator, the adversary h is trained to recover p from the latent code x' generated by f, this leads to minimizing the discriminator loss:

$$\mathcal{L}_{\text{disc}} = \text{MSE}(p, \hat{p}) = \|p - (h \circ f)(x)\|^2, \tag{3.4}$$

   note that in this phase the encoder and decoder will be frozen since we only want to update the discriminator.

   In the case when p is a categorical variables, $\mathcal{L}_{\text{total}}$ and $\mathcal{L}_{\text{disc}}$ will be adjusted to use binary cross entropy loss between p and $\hat{p}$.

The adversarial interaction between f and h drives the encoder to strip x' of any predictive signal about p, while still retaining the information needed for g to reconstruct x. When the system reaches equilibrium, it indicates that the decoder g can accurately reconstruct x given (p, x'), and the adversary h cannot recover p from x', signaling that disentanglement has been achieved. Furthermore, Algorithm 1 outlines the adversarial training procedure implemented in Python. In the upcoming sections, figures such as Figure 3.6 illustrates how the losses of this AAE training procedure reach equilibrium.

Algorithm 1 Training Process of the Adversarial Autoencoder

for each batch (x, p) **do**
  Phase 1: Train Encoder and Decoder
  Set f and g to train mode. Freeze h
  Zero gradients of autoencoder
  $x' \leftarrow f(x)$
  $\hat{x} \leftarrow g(x', p)$
  $\hat{p} \leftarrow h(x')$
  $\mathcal{L}_{\text{recon}} \leftarrow \text{MSE}(\hat{x}, x)$
  $\mathcal{L}_{\text{disc}} \leftarrow \text{Loss}(\hat{p}, p)$
  $\mathcal{L}_{\text{total}} \leftarrow \mathcal{L}_{\text{rec}} - \beta\mathcal{L}_{\text{disc}}$
  Backpropagate and update f and g
  Step 2: Train Discriminator h
  Set h to train mode. Freeze f and g
  $\hat{p} \leftarrow h(x')$
  $\mathcal{L}_{\text{disc}} \leftarrow \text{MSE}(\hat{p}, p)$
  Backpropagate and update h
end for

3.3.3 Auditing Procedure to Measure Indirect Influence

Step by step, we are building the foundation blocks of the framework. Until now, in the previous chapter, we examined how SHAP values can be utilized to measure direct influence of features on a model's prediction. In this chapter, we covered the Disentangling Influence Audits method that reduces the problem of measuring indirect influence into a problem of computing the direct influence of the disentangled representations. In the preceding sections, we defined the concept of disentangled representations are provided detailed explanations of the procedure for constructing them using AAEs. We now proceed to the final components of the method; how to audit indirect influence using the disentangled representations.

Given a model (either original model or a black-box) with input x, the direct influence of a feature p, ϕ_p, is defined by equation (2.4) with a slight modification in the computation method of the Marginal Contribution:

$$\phi_p(M, x) = \sum_{z \subseteq x} \frac{|z|!\,(n - |z| - 1)!}{n!} \left[M_x(z) - M_x(z \backslash p)\right], \tag{3.5}$$

where n is the number of features, $|z|$ denotes the number of nonzero entries in z, $z \subseteq x$ is a vector whose nonzero entries are a subset of the nonzero entries in x. Moreover, $z \backslash p$ denotes z with the feature p set to zero. Lastly, $M_x(z) = \mathbb{E}[M(z)|z_S]$ is the conditional expected value of the model given to fixing all the nonzero entries of z (S is the set of nonzero entries in z).

Marx et al. also propose the following definition for computing indirect influence via reduction to indirect influence over the disentangled Model:

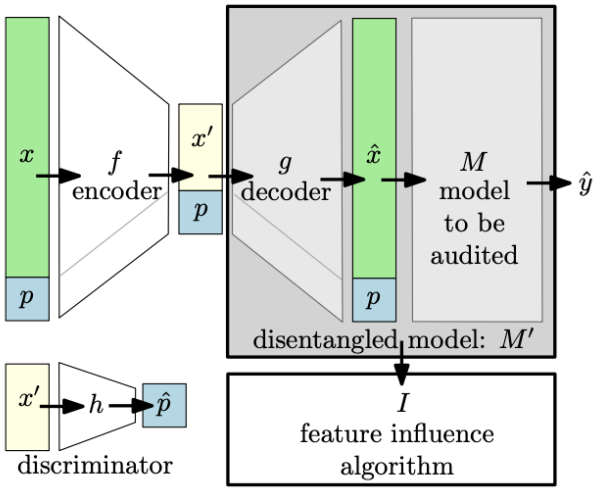


Figure 3.4: Architecture for auditing the indirect influence of a feature p on a model's M predictions for the input data (p, x). The autoencoder $g \circ f$ reconstructs the input by learning a latent representation (p, x') where x' is independent of p. This is enforced by the adversary h. The indirect influence audit will be applied on the disentangled model $M' = M \circ g$, which receives the disentangled representation (p, x) as input and is audited using a direct influence method. This figure is taken from Marx et al. [21].

Definition 3.1 (Indirect Influence)**.** *The indirect influence of a feature p on the prediction of a model M(p, x) is the direct influence ϕ_p of p on the prediction of the disentangled model ($M \circ f^{-1}$)(p, x′):*

$$\mathcal{II}(M, (p, x)) = \phi_p(M \circ f^{-1}(p, x')). \tag{3.6}$$

In equation (3.6), the indirect influence of p on the model M will be instead computed as the direct influence of p on M when the model operates on the disentangled representations. This is what is referred to as the disentangled model, later denoted by M', and it is implemented as follows:

- The disentangled model M' takes in the disentangled representations (p, x').
- The decoder g then reconstructs the original input, $\hat{x} = g(p, x')$.
- The reconstructed input $\hat{x}$ is then passed through the model M, yielding the final prediction $\hat{y} = M(\hat{x})$.

In this setup, $M' = M \circ g$. Note that learning an invertible mapping f from (p, x) to (p, x') may not be possible in practice. However, we can use g (the decoder) as an approximation. It is also specified that suppose for a set S containing features that may or may not be proxies of p, and there is some function W that can predict p from a subset of the features x_S (i.e. $W(x_S) \approx p$ (in this case this function can be thought of as the discriminator), if there are no features that can predict p, then the indirect and direct influence of p are the same as the only proxy for p is itself [21].

Algorithm 2 Disentangled Influence Audit (X, M)

Require: The input data X, and the model M.
Require: The encoder f, the decoder g, and the discriminator h.
Require: The Disentangled_Model $M' = M \circ g$.
1: **for** p **in** FEATURES(X) **do**
2: Train $f, g,$ and h.
3: $X' \leftarrow f(X)$ **// Get the latent code**
4: dis_rep $\leftarrow (X', p)$ **// Create the Disentangled representations**
5: $\text{SHAP}_p \leftarrow$ Direct_Influence(M', dis_rep) **// Compute ϕ_p on M'**
6: **end for**
7: **return** $\{\text{SHAP}_p\}$

3.3.4 Implementation of the Auditing Procedure

To implement the Disentangled Influence Audit method, we follow a structured procedure similar to the Algorithm of Marx et al. [21] as demonstrated in Algorithm 2:
Algorithm 2 outlines the overall process of computing the indirect influence of p. Prior to applying the algorithm 2, the necessary components should be defined: the encoder, the decoder, the discriminator, the disentangled model, and a function that creates disentangled representations (X', p) (optional). Additionally, the input data X used may be the whole dataset or mini-batches (depending on user preferences).

Once the requirements are ready, the process begins. For every feature p that we want to compute the indirect influence of, f, g, and h will be trained each time. Once trained, the latent representation $X' = f(X)$ will be produced, from which the disentangled representation (X', p) will be formed. Next, this will be passed through the disentangled model $M'$for analysis. To quantify the indirect influence of p, the direct influence function from the SHAP Python package [17] will be applied to M' and the input (X', p).

As discussed and observed from Algorithm 2, the process will be repeated for every feature of interest p. While compared to other methods, this process is simpler and more effective, it could be more computationally expensive for large datasets. Hence, Marx et al. recommend that it could be more cost-efficient to apply the method for specific features of particular concern.

3.4 Numerical Experiments

In this section, we aim to put into practice the theoretical concepts explored in this chapter regarding the auditing of machine learning models and measuring the indirect influence of features on the model's predictions through different numerical experiments (from synthetic to real-world problems). By making associations with Section 2.5, we will observe how features that might not exhibit direct influence on a model's output, actually affect the predictions indirectly. Visualizations of the direct and the indirect influences provides a

deeper view into the behavior of the (black-box) model. By the end of this section, we will have a clearer picture of how the models work behind closed doors, providing a way to ensure fairness, transparency, and reliability.

3.4.1 Synthetic Experiments

Before attempting to measure any indirect influence of features, our first objective was to develop the model from scratch and reproduce the results of the synthetic experiment $x + y$ carried out by Marx et al. [21] (dataset and model information can be found in Section 2.5.1). We begin with this experiment because not only is it a simple experiment but it would also allow us to establish a solid foundation on how to approach these problems for more complex datasets.

We observed in Section 2.5 (Figure 2.5), only the features x and y showed direct influence on the model's predictions. In this controlled setting, where the contributions of each of the features x, $2x$, x^2, y, $2y$, y^2, z, $2z$, and z^2 are clearly evident, Marx et al. performed two additional experiments on indirect influence; with the aim of examining the quality of the disentangled representations and ensuring that the Disentangled Influence Audit approach accurately captures all the influence of individual-level features on the outcome. To do so, they considered the following two different models to produce disentangled representations:

- A handcrafted disentangled representation; where nine separate models (i.e. for each feature) are handcrafted to perfectly disentangle each of the nine features.
- A learned disentangled representation; the models (the Encoder, the Decoder, and the Discriminator) are created according to the adversarial autoencoder training process as discussed in Section 3.3.4.

In this section, we conduct the following experiments:

- Use the exact model for $x+y$ (as mentioned in Section 2.5.1), and handcrafted models (i.e. explicit formulas) for the encoder, the decoder.
- Use the exact model for $x + y$, and learned disentangled representations.

Case1: Exact Model With Handcrafted Models for the Encoder and Decoder: Following the set up of the handcrafted models from their supplementary papers, we constructed the following explicit formulas for the encoder, and the decoder:

Denote p as the protected feature that we want to compute the influence of, and consider X to be all the features excluding p. Since we want an encoder function $f(X, p)=(X', p)$ that outputs the latent representation X' independently of p. This gives us the following:

- We construct f that produces X' based on the conditions that X' should not retain any information about p. We have the following three options for X':

$$X' = \begin{cases} (x, z) & \text{if } p \in \{y, 2y, y^2\} \\ (y, z) & \text{if } p \in \{x, 2x, x^2\} \\ (x, y) & \text{if } p \in \{z, 2z, z^2\} \end{cases} .$$

- We construct the decoder function g so that it maps (X', p) back to the original feature vector $\{x,\ 2x,\ x^2,\ y,\ 2y,\ y^2,\ z,\ 2z, z^2\ \}$. Marx et al. mention that, for instance $p = y^2$, the decoder reconstructs the original feature vector by first computing $\sqrt{y}$ to obtain y from the protected feature, then uses this to compute $2 \cdot y$. All the other features relating to x and z are computed naturally (that is, to get $2x$, multiply x with 2, etc.). Let R denote this sub-step function to recompute the original features, and v denote any arbitrary variable, we obtain the following:

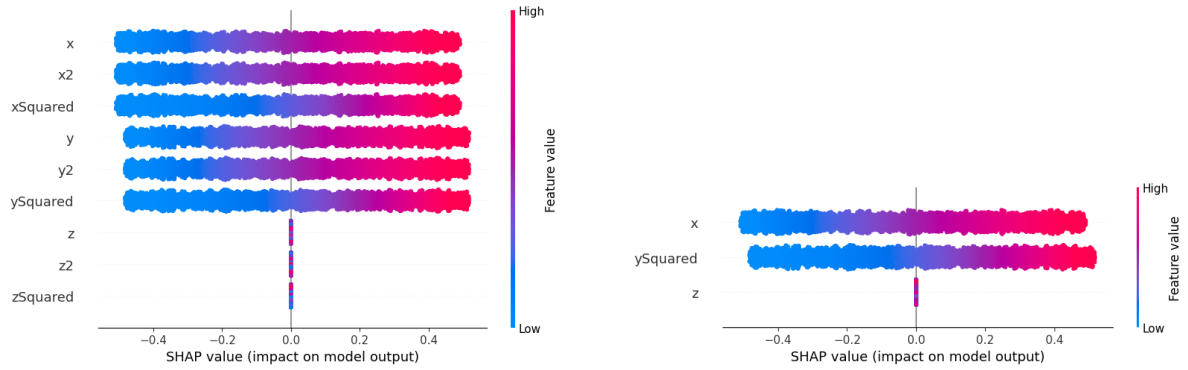
$$R(v) = \begin{cases} (v, 2 \cdot v, v^2) & \text{if } v \in \{x, y, z\} \\ (\frac{v}{2}, v, (\frac{v}{2})^2) & \text{if } v \in \{2x, 2y, 2z\} \\ (\sqrt{v}, 2\sqrt{v}, v) & \text{if } v \in \{x^2, y^2, z^2\} \end{cases}.$$

- The decoder function g will output the reconstruction $(\hat{X}, p)$:

$$(\hat{X}, p) = \begin{cases} (R(X'_1),\ R(p),\ R(X'_2)) & \text{if } p \in \{y, 2y, y^2\} \\ (R(p),\ R(X'_1),\ R(X'_2)) & \text{if } p \in \{x, 2x, x^2\} \\ (R(X'_1),\ R(X'_2),\ R(p)) & \text{if } p \in \{z, 2z, z^2\} \end{cases},$$

  where X'_i denotes to the feature at index i for $i \in \{1, 2\}$.

Now that the encoder and decoder are constructed, using the same process as in Algorithm 2 (where the encoder and the decoder are explicit functions mentioned above. No decoder in the handcrafted version), we can apply the procedure to measure the indirect influences. Figure 3.5 represents the results of our experiments.



(a) (b)

Figure 3.5: This figure presents the results of measuring the indirect influence of the features using the exact model of $x + y$ with the handcrafted explicit formulas for the encoder and the decoder. Figure 3.5a displays that all the features relating to x and y (i.e. the proxy variables) are indirectly impacting the model's output. Since the auditing reduces the process of measuring indirect influence of a feature into computing its direct influence on the disentangled model, Figure 3.5b shows the direct influence of the protected feature (in this case y^2) on the disentangled model. This is the step where the direct influence of p is saved, process is repeated until all the features of interest are evaluated.

By only looking at Figure 2.5b, one would assume that only x and y have a role in influencing the predictions of the model. However, Figure 3.5a tells us a completely different story; although the proxy variables $2x$, x^2, $2y$, and $y^2$are not directly used by the model (and not obvious on the surface), they are indirectly affecting the model's outputs.

Suppose we wanted to analyze the auditing process of only one of the features, consider $p = y^2$. Since computing the indirect influence of a feature on a model's output is reduced to simply computing the direct influence of the feature on the disentangled model, we provide Figure 3.5b to show the background operations done for deeper understanding. Noting that the disentangled model takes input (X', p), from this figure, we observe the direct influences of the features x, z, andy^2 on the disentangled model. Since our feature of interest is y^2, we save its corresponding SHAP values and repeat the process for all the features that we want to compute the influence of. In the end, making a complete dataset of all these SHAP values, we are able to produce the results in Figure 3.5a.

Case2: Exact Model With Learned Disentangled Representations: In this section, we present the results of applying the Disentangled Influence Audit procedure using a learned encoder, decoder, and discriminator on the exact model $x + y$.

We follow a similar process as Marx et al. in building our encoder, decoder, and discriminator; they are trained to learn disentangled representations, i.e., learn to produce a latent code x' that is independent of the protected feature p through the adversarial training process. Keeping the overall architecture the same, each model contains two hidden layers with 10 units each and ReLU activations. The final layer of both the decoder and the discriminator uses a sigmoid activation. As in their setup, we use a 4-dimensional latent vector, set $\beta = 0.5$,and train with the Stochastic Gradient Descent (SGD) optimizer for $100,000$ epochs using a fixed learning rate of 0.01. It should also be noted that there is a mismatch between their theory and in the implementation of learned disentangled representations; the input x is considered to be all features, including p. Nonetheless, x' will be independent of p.

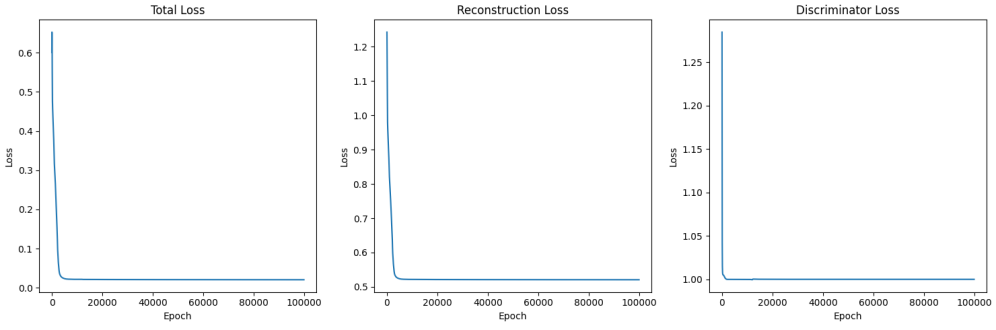


Figure 3.6: This figure shows the total loss, reconstruction loss, and discriminator loss incurred by the encoder, decoder, and discriminator during the training process. The results in this figure correspond to the case where the protected feature is z. However, all the other features obtained similar results.

To better understand the training process and to verify the correctness of the implementa-

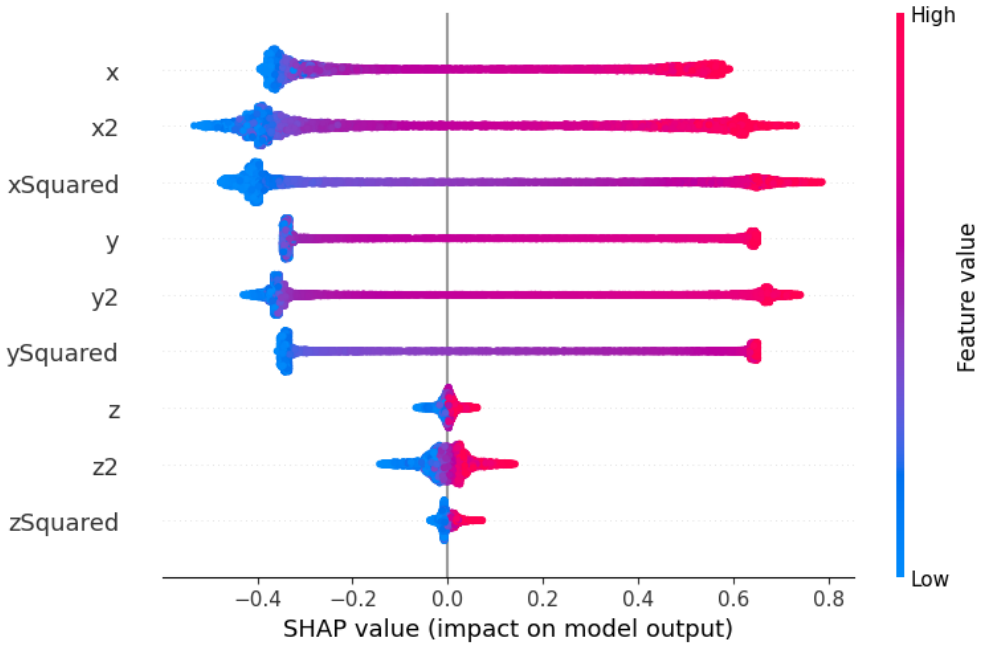


Figure 3.7: This figure illustrates how indirect feature influence is detected using Adversarial Autoencoders applied to the exact model $x + y$ with disentangled representations learned through adversarial training. The noise terms have do not exhibit any indirect influence, while all the features relating to x and y do.

tion, we plotted the training losses over $100,000$ epochs, including total loss, reconstruction loss, and discriminator loss. Since the models are trained for every protected feature, Figure 3.6 displays the losses for the case $p = z$.

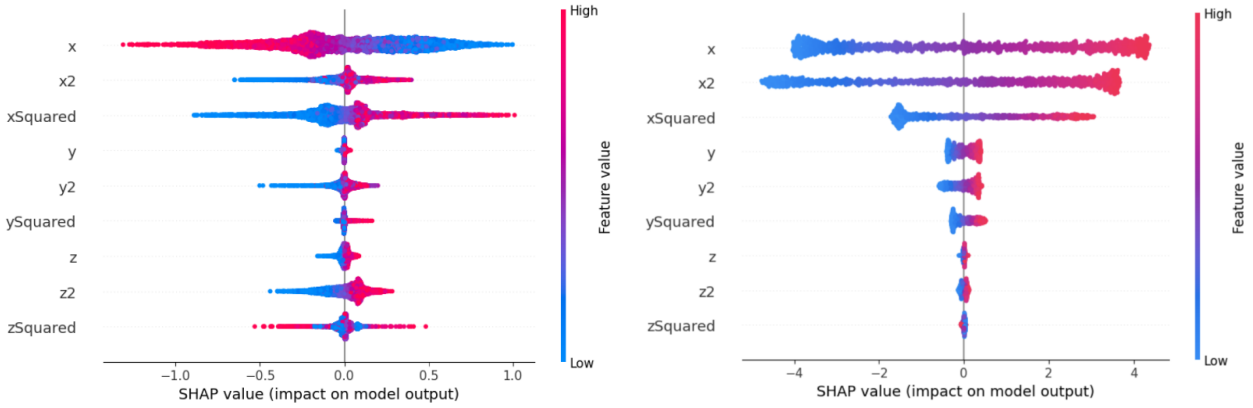
From this figure, we observe that the discriminator loss is converging to 1. At first glance, this might suggest poor performance, however, in the context of adversarial training, this behavior is desirable. Although the objective of the discriminator is to be able to predict the protected feature p from the latent representation x' by minimizing the loss between the predicted p and the true p, in the adversarial setting a loss near 1 indicates that discriminator is not able to correctly predict p from x' i.e. at the equilibrium this implies that x' has become independent of p.

Simultaneously, the total loss in Figure 3.6 steadily decreases and plateaus near 0, indicating that the model is learning a latent representations x' that retains important information from the data for reconstruction, while making sure that x' does not contain predictive information about p. Although having a reconstruction loss that converges to 0.5 may not seem optimal, it reflects the trade-off between the reconstruction fidelity and the adversarial training purpose. Moreover, from the results in Figure 3.7 we can conclude that the AAE successfully learned disentangled representations and detected indirect influence. Similar to the results of the handcrafted case in Figure 3.5, the AAE is able to capture the influence of the proxy variables while attributing negligible importance to the noise terms.

To observe the direct influence of the disentangled representations (x', p) on the disentangled model, refer to Figure A.1 in the Appendix. When analyzing, unlike Figure 3.5b, the beeswarm plots of this will not have feature names since we do not know what they are; however, due to the composition process of (x', p), p is indexed at the last position, allowing us to obtain the direct influence of the protected feature on the disentangled model.

Indirect Influence of the Algebraic Operation $10x + y$

Interestingly, when the same model, and same procedure was used to measure the indirect influences of the exact model that computes $10x + y$, the results might seem slightly unexpected. Figure 3.8a shows the results when the AAE was trained without batching, while Figure 3.8b presents the results when the AAE was trained with batches of size 200. When trained with batches, it is able to better identify the indirect influence of the features, unlike in Figure 3.8a which shows that the model captured indirect influence for the noise terms $z, z2,$ and zSquared. These observations indicate that batch training improves the model's ability to detect indirect influences.



(a) (b)

Figure 3.8: Results of the Disentangling Influence Audit method applied to the exact model that computes $10x + y$. Figure 3.8a is without batching, while Figure 3.8b is with batch size of 200.

3.4.2 Adult Income dataset

In Section 2.5.3, we analyzed the direct influence of features on the model predictions using SHAP beeswarm plot. Several features showed significant global impact, while some features showed little to no direct influence. To explore more and to provide deeper insights, in this Section we focus on analyzing the SHAP beeswarm plot in the context of the indirect influence of features.

Using the same data, predictive model, and preprocessing steps as in Section 2.5.3, we applied the Disentangling Influence Audit procedure to assess the indirect influence of features.

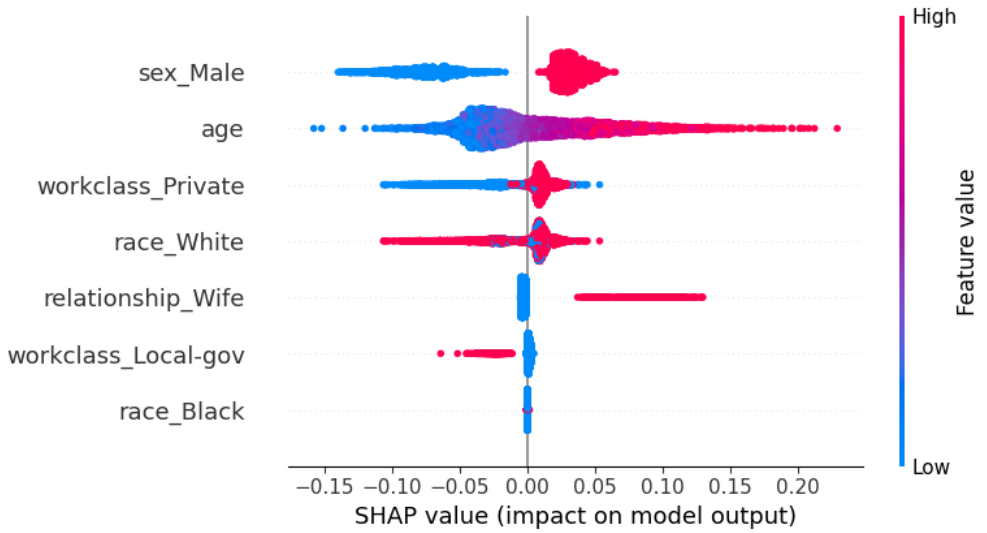


Figure 3.9: SHAP beeswarm plot of the Adult Income dataset presenting the results of applying the Disentangled Influence Audit procedure for capturing the indirect influence of seven chose features.

Details on the architecture of the encoder, decoder, discriminator, and training process are specified in the Appendix A.2.1. Before describing the obtained results, we should note that since this is a dataset with 42 features, the process of training the AAE can be computationally expensive. Instead of applying the Disentangling Influence Audit procedure on all the features, we focus on a subset of features where some appeared less influential. As a result, Figure 3.9 SHAP beeswarm plot of the indirect influence of only seven features. These features include: Male, Age, Workclass Private, Race White, Relationship Wife, Workclass Local Government, and Race Black. Age is a numerical variable, while the other features are all categorical that have been one-hot encoded. This is why their format is "feature name__category." For instance, workclass__private means the individual belongs to the "Private" category of the workclass variable. Low values in Figure 3.9 indicate that a one-hot encoded feature is False. Additionally, the objective of this experiment is to not only evaluate the indirect influences of these features, but to also verify that our model is effectively capturing these influences. We will compare our results with those of Marx et al. as a baseline for validation. It is important to note that differences in training, model architectures may contribute to the differences in the results.

Looking back at Figure 2.12, it is evident that the features Male, Workclass Private, and Race White displayed no direct influence on the model predictions. However, as observed in Figure 3.9, our analysis reveals a different outcome, indicating that these features, in fact, have an impact on the model's behavior indirectly. Whereas the age feature has not changed. Furthermore, we also provide the training losses of the encoder, the decoder, and the discriminator over the epochs, which offer insight into the AAE's process of learning disentangled representations. The loss curves allow us to analyze the model's convergence, and evaluate the quality of the adversarial training of the encoder and the discriminator. As displayed

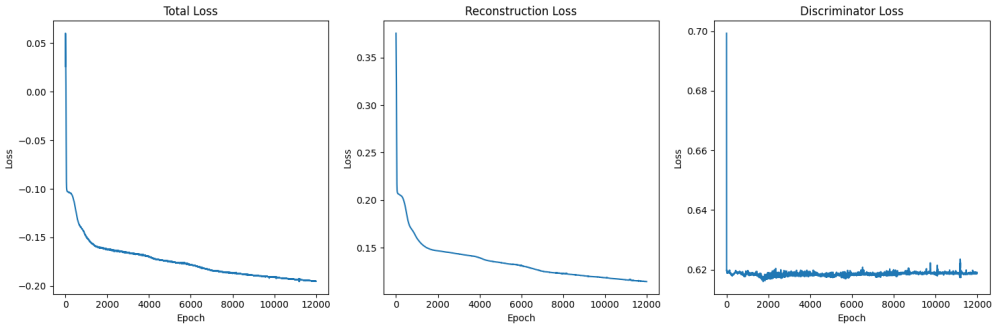


Figure 3.10: This plot presents the three losses: the total loss, reconstruction loss, and discriminator loss of the trained Adversarial Autoencoder for the Adult Income dataset over 12000 epochs. This is for when the protected feature is considered to be the Workclass Private type.

in Figure 3.10, the decrease in the reconstruction error informs us that the autoencoder is effectively learning to reconstruct the original input from the latent representation, and the convergence of the discriminator error to approximately 0.6 indicates that the discriminator is not being successful in predicting the protected feature from the latent representation x'. On the other hand, the total loss going to -0.2 indicates that discriminator is more dominant than the the reconstruction, which can be verified by subtracting half of the discriminator loss from the reconstruction from the plots. This signals that the encoder is successfully producing latent representation x' that retains the necessary important information for reconstruction, while ensuring that x' is independent of the protected feature p. These losses align with the detected indirect influence, further reinforcing the effectiveness of the auditing procedure the indirect influence of the variable Workclass Private.

One difference between our indirect influences in Figure 3.10, and those of Marx et al. is that their model uncovered influence for the feature Race Black, while ours did not. This discrepancy could occur due to several factors as previously discussed, and it simply indicates that different tuning is needed for more accurate results.

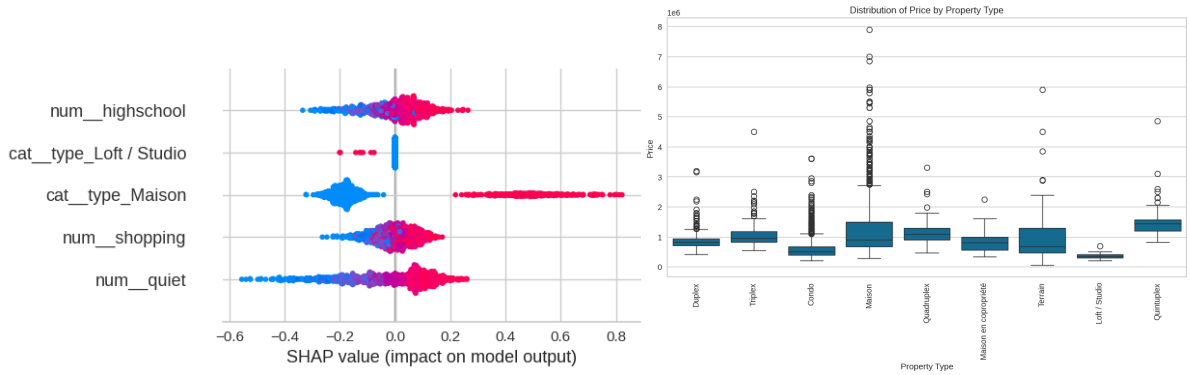
3.4.3 Montréal Housing Dataset

Using the same dataset, and the trained Random Forest Regressor model as in Section as in Section 2.5.4, we decided to audit the model to detect if any of the input features indirectly impact the predictions on average. Note that since computing the indirect influence of features is more computationally expensive, we decided to apply it on five features. The property types Loft/Studio and Maison (i.e., house), and the Local Logic scores high school, shopping, and quiet.

Recall from Figure 2.13 that the features superficie du terrain, price per square feet, and pieces beta were the only features that had a direct influence on model predictions. However, Figure 3.11a shows that after implementing the Disentangling Influence Audit method on the data, it was found that the features that initially did not have direct influence are, in

fact, indirectly influencing the model's predictions.

From Figure 3.11a, observe that the property type Loft/Studio has a negative impact on the predicted price. This means that, on average, the property type is a Loft or a Studio is associated with lower predicted prices, whereas if the property type is Maison then the predicted price tends to increase. These results are consistent with the price distribution by property type shown in Figure 3.11b, where lofts and studios are priced lower compared to houses. Another interesting observation is the result of the high school score, which shows that houses near high schools tend to have higher prices. This pattern aligns with the preference of families to live in neighborhoods where schools are more easily accessible. The



(a) (b)

Figure 3.11: Figure 3.11a displays SHAP beeswarm plot of the indirect feature influences of the Montreal Housing Dataset. Only five features were considered due to computational complexity: The property types Loft/Studio and Maison, and the Local Logic scores high school, shopping, and quiet. Figure 3.11b shows the price distribution by property type. The y-axis is in millions.

primary goal of this experiment was to evaluate how effectively the method identifies the indirect influence of features, and the results are promising. However, for a deeper analysis of such datasets, it is also important to explore the dependence plots to better understand the interactions between features and their combined effects on the model's predictions.

Chapter 4

Conclusion

Ensuring transparency and reliability of AI and machine learning models demands a rigorous understanding of how and why ML models such as neural networks make their decisions. In this thesis, we examined several methods proposed with the aim of explaining black-box models; the primary techniques employed in the conducted experiments were SHAP values for direct influence, and the Disentangled Influence Audit for indirect influence. We studied how Marx et al. break down the problem of measuring indirect influence, into a problem of measuring direct influence of disentangled representations. We provided in-depth explanations of all the necessary background information needed to not only have a strong foundation when introducing the main concepts, but to also be able to interpret the SHAP plots.

Our key contributions include implementing these methods on several experiments from synthetic experiments to real-world applications. The aim of the experiments was to ensure correct application of the studied methods. Hence, to test if the models that we built were performing well, we reproduced two key experiments (the numerical synthetic experiment, and the Adult Income classification) by Marx et al. We conducted additional experiments where we applied the Disentangling Influence Audit method on the $10x + y$ model and the Montréal Housing data, demonstrating the method's ability to identify the indirect influence of the features that exhibited no direct influence. Lastly, we also contributed by preforming convergence analysis of the SHAP in the direct influence environment. We obtained that the estimated SHAP value converged to the true value with an order of convergence $O\left(\frac{1}{\sqrt{n}}\right)$ by observing the log-log plot of the average absolute error and the sequences.

While our architectural choices closely mirror those of Marx et al., there is room to experiment with alternative architectures and regularization techniques such as sparse regularization and sparse autoencoders to examine if they affect the quality of the disentangled representations, and provide more efficient results. Another open research direction is the need to find better datasets related to Montréal housing. Possibly, we need datasets that include information about the properties and the individuals living in them. However, such sensitive data might be hard to obtain.

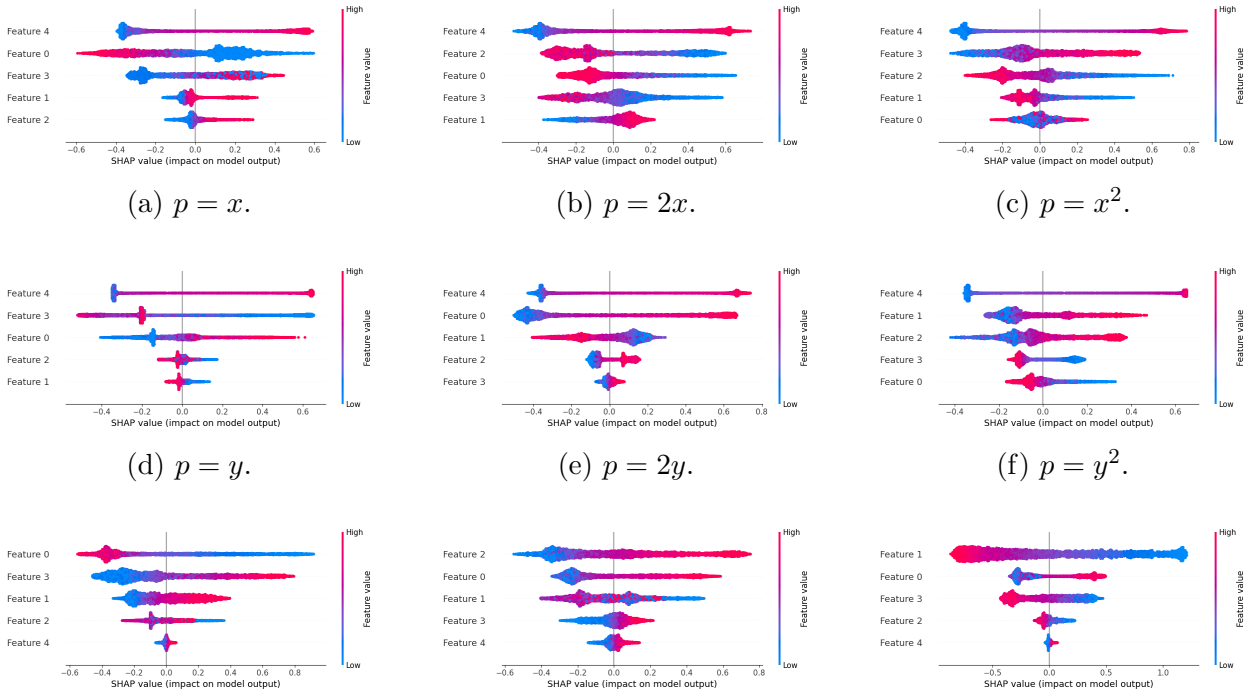
In summary, this thesis highlights the effectiveness of the Disentangled Influence Audit method in detecting indirect influences in both synthetic and real-world datasets and provides a foundation for future work on model interpretability, data acquisition, and practical auditing applications.

Appendix A

Additional Details

A.1 Synthetic Experiment

Exact Model with Learned Disentangled Representation The following figure provides a deeper view into the computation process of the indirect influence of the protected features on the model's outputs; which is computed as the direct influence of the disentangled representations (x', p) on the disentangled model. Due to the formatting of (x', p), p is placed at the last index of the disentangled representation vector. Hence, when observing Figure A.1, feature 4 represents the direct influence of p on the disentangled model.



(a) $p = x$. (b) $p = 2x$. (c) $p = x^2$.

(d) $p = y$. (e) $p = 2y$. (f) $p = y^2$.

(g) $p = z$. (h) $p = 2z$. (i) $p = z^2$.

Figure A.1: Visualizations of the direct influence of the protected feature p (feature 4) on the disentangled model.

A.2 Adult Income Dataset

A.2.1 Data Preprocessing and Model Architecture

In this section, we outline the data preprocessing steps and the model used to classify whether an individual makes an annual income of more than 50k. All preprocessing steps follow the methodology of Marx et al. [21].

- Categorical features are:
  - One-hot-encoded using the pandas get_dummies function.
  - Categorical values that have less than 1000 occurrences are binned into a category "rare_value."
- Numerical features are normalized to have mean 0 and standard deviation of 1.
- The feature "Educ-Num" is dropped.

The Classifier Model Marx et al. train a classifier for predicting whether an individual makes an income of 50k label with binary cross entropy loss and no hidden layers. They obtain a test loss of 0.326 and a test accuracy of 84.9%. We also train a Logistic Regression model, and obtain the same exact losses.

Architecture of the Encoder, Decoder, and Discriminator To produce disentangled representations and to measure the indirect influence of the features, Marx et al. use the following model architectures for the encoder, decoder, and discriminator:

- The encoder, decoder, and discriminator have 2 hidden layers: layer 1 has 25 hidden units, and layer 2 has 12 hidden units.
- Dimension of the latent vector is 10.
- Value of the hyperparameter $\beta = 0.5$ as the importance of disentanglement for the encoder.
- The models are trained for 12000 train steps with minibatch sizes of 16, using SGD with a constant learning rate of 0.01.
- Data is split into 80%train and 20%test set for training.

A.3 Montréal Housing Dataset

Table A.1 provides a summary of the data features with their descriptions.

Features	Description
Type	Categorical: Condo, Maison, Duplex, Triplex, Quadruplex, Quintuplex, Maison en copropriété, Terrain, and Loft / Studio.
Quartier	Categorical: Peter McGill, Le Plateau-Mont-Royal, Saint-Laurent, Petite-Bourgogne, Vieux-Montréal, Faubourg Saint-Laurent, Rosemont, LaSalle, Côte-des-Neiges, Nord-Ouest de lIle de Montréal, Centre-Sud, Ahuntsic, Hochelaga-Maisonneuve, Notre-Dame-de-Grâce, Pointe-aux-Trembles, Mercier-Est, Lachine, Petite-Patrie, Bordeaux-Cartierville, Villeray, Rivière-des-Prairies, Montréal-Nord, Mercier-Ouest, Saint-Michel, Anjou, Outremont, Verdun, Saint-Léonard, Pointe Saint-Charles, Ville-Émard/Côte Saint-Paul, Saint-Henri, Parc-Extension, and Autre.
superficie du terrain	Numerical: Land area.
pieces_beta	Numerical: total number of rooms.
Local Logic scores (18 features)	Numerical: scores on a scale of $0 - 5$. The 18 different scores are: high-school, primary_school, transit, groceries, wellness, resto, pedestrian, green, historic, cycling, car, vibrant, shopping, daycare, nightlife, cafe, quiet, and parks.
price	Numerical: in dollars. This is the target feature. In the training log transformation will be applied on the price.

Table A.1: Summary of the Montréal Housing data features.

A.4 Neural Networks and Rectified Linear Unit (ReLU)

The Rectified Linear Unit (ReLU) is a very commonly used activation function in neural networks due to its simplicity and computational efficiency. It is defined as:

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

That is, given an scalar $x \in \mathbb{R}$, if it is positive, then the function returns the input itself, or 0 otherwise:

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

ReLU allows neural networks to learn nonlinear patterns in the data and solves the vanishing gradient problem that affects other activation functions such as the sigmoid or tanh.

Neural Networks Artificial neural networks (NNs) are a class of machine learning models inspired by the human brain. Similar to the human brain, NNs have neurons to process information and learn patterns from the data. Mathematically, they can be thought of as functions (composition of affine functions and activation functions).

Figure A.3 presents the architecture of a neural network. A basic neural network consists of layers of neurons, which are interconnected units. First layer is called the input layer, the last layer is the output layer, and the layers between the input and output layers are called the hidden layers. Each neuron computes a weighted sum of the inputs it receives followed by a nonlinear activation function.

For a single layer neural network with one neuron (Figure A.2), the output is expressed as follows:

$$\mathbf{y} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}),$$

where $\mathbf{x} \in \mathbb{R}^n$ is the input vector, $\mathbf{W} \in \mathbb{R}^{m \times n}$ is the weight matrix, $\mathbf{b} \in \mathbb{R}^m$ is the bias vector, σ is an element-wise activation function (such as, ReLU, sigmoid), and $\mathbf{y} \in \mathbb{R}^m$ is the output of the layer.

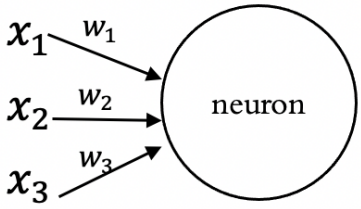


Figure A.2: A single layer neural network with one neuron. For instance, the input vector is $[x_1, x_2, x_3]$, and the weight vector is $[w_1, w_2, w_3]$. The neuron computes a weighted sum of the inputs it receives followed by a nonlinear activation function (if there is any).

.

For a multilayer neural network, the output of each hidden layer $\mathbf{h}_i \in \mathbb{R}^{N_i}$ where $i = 1, \cdots, L-1$ is:

$$\mathbf{h}_i = \sigma(\mathbf{W}_i \cdot \mathbf{h}_{i-1} + \mathbf{b}_i),$$

where

- L: number of layers (hidden + output).
- N_i: number of neurons in each hidden layer.
- $\mathbf{h}_0 = \mathbf{x} \in \mathbb{R}^d$.
- $W_i \in \mathbb{R}^{N_i \times N_{i-1}}$ and $b_i \in \mathbb{R}^{N_i}$.
- $\sigma : \mathbb{R} \to \mathbb{R}$ is an activation function.

The output is:

$$f(\mathbf{x}) = \mathbf{W}_L \cdot h_{L-1} + \mathbf{b}_L.$$

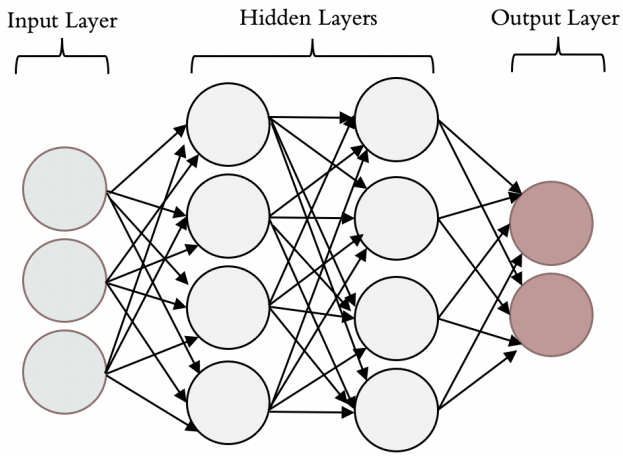


Figure A.3: Architecture of a neural network, which consists of layers of neurons. First layer is the input layer, the last layer is the output layer, and the layers between the input and output layers are the hidden layers.

Bibliography

[1] Philip Adler, Casey Falk, Sorelle A Friedler, Tionney Nix, Gabriel Rybeck, Carlos Scheidegger, Brandon Smith, and Suresh Venkatasubramanian. Auditing black-box models for indirect influence. *Knowledge and Information Systems*, 54:95–122, 2018.

[2] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS One*, 10(7):e0130140, 2015.

[3] Barry Becker and Ronny Kohavi. Adult. UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5XW20.

[4] Simone Brugiapaglia. Numerical analysis notebooks: Convergence. `https://github.com/simone-brugiapaglia/numerical-analysis-notebooks/blob/main/02_convergence.ipynb`, 2024.

[5] Richard L. Burden and J. Douglas Faires. *Numerical Analysis*. Brooks/Cole, Cengage Learning, 9 edition, 2010.

[6] Centris. Real estate listings in quebec. `https://www.centris.ca/`, 2025.

[7] Joymallya Chakraborty, Suvodeep Majumder, and Tim Menzies. Bias in machine learning software: Why? how? what to do? In *Proceedings of the 29th ACM joint meeting on European Software Engineering Conference and Symposium on the Foundations of software Engineering*, pages 429–440, 2021.

[8] Anupam Datta, Shayak Sen, and Yair Zick. Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems. In *2016 IEEE symposium on security and privacy (SP)*, pages 598–617. IEEE, 2016.

[9] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[10] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, et al. *An introduction to statistical learning*, volume 112. Springer, 2013.

[11] Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[12] G Kutyonik. The mathematics of reliable artificial intelligence. *SIAM News, July/August*, 2024.

[13] Jeff Larson, Surya Mattu, Lauren Kirchner, and Julia Angwin. Machine bias: There's software used across the country to predict future criminals. and it's biased against blacks. *ProPublica*, 2016.

[14] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[15] Stan Lipovetsky and Michael Conklin. Analysis of regression in game theory approach. *Applied Stochastic Models in Business and Industry*, 17(4):319–330, 2001.

[16] Scott Lundberg and Su-In Lee. Shap: Shapley additive explanations. `https://shap.readthedocs.io/`, 2021.

[17] Scott Lundberg and Su-In Lee. Shap: Shapley additive explanations, 2021.

[18] Scott Lundberg and Su-In Lee. adult.data. `https://github.com/slundberg/shap/blob/master/data/adult.data`, 2025.

[19] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 30, 2017.

[20] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

[21] Charles Marx, Richard Phillips, Sorelle Friedler, Carlos Scheidegger, and Suresh Venkatasubramanian. Disentangling influence: Using disentangled representations to audit model predictions. *Advances in Neural Information Processing Systems*, 32, 2019.

[22] Ziad Obermeyer, Brian Powers, Christine Vogeli, and Sendhil Mullainathan. Dissecting racial bias in an algorithm used to manage the health of populations. *Science*, 366(6464):447–453, 2019.

[23] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144, 2016.

[24] Lloyd S Shapley et al. A value for n-person games. 1953.

[25] Avanti Shrikumar, Peyton Greenside, Anna Shcherbina, and Anshul Kundaje. Not just a black box: Learning important features through propagating activation differences. *arXiv preprint arXiv:1605.01713*, 2016.

[26] Erik Štrumbelj and Igor Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and Information Systems*, 41:647–665, 2014.

[27] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

[28] Guijuan Zhang, Yang Liu, and Xiaoning Jin. A survey of autoencoder-based recommender systems. *Frontiers of Computer Science*, 14:430–450, 2020.