Bachelor's Thesis

DEGREE IN MATHEMATICS

EXPLAINABLE MACHINE LEARNING AND CREDIT RISK MODELS

Addressing Explainability with Cooperative Games Theory and an Application to Credit Risk Assessments with Neural Networks

Aleix Sancho Jiménez

Supervised by Dr./Prof. Alejandra Cabaña Nigro

June 2024

UAB
Universitat Autònoma de Barcelona

Any sufficiently advanced technology is
indistinguishable from magic.
Arthur C. Clarke

Preface

Until the trick gets revealed for most of us magic is related with the unknown and the unpredictable. Thus, it produces an instinctive reaction of precaution, alertness and curiosity. As an universal law, new advancements in science neither escape this initial fate and Explainable Machine Learning is just another proof for that fact.

Along with more powerful and sophisticated ML models comes both attraction and skepticism from non-IA-expert fields. Therefore, the demand from techniques that help to bring light to their inherent opaqueness is bigger than ever. Explainable Machine Learning – XML from now on – tries effectively to answer to the question of what to do when the model is so complex that we cannot comprehend it and the purpose here is to present clearly and concisely some of its great achievements.

We will start by introducing the fundamentals of XML and the main techniques that we will explain: LIME, DeepLIFT and Shapley Values. Later, we will notice the essence of these techniques to link them with Cooperative Games Theory and unify them all such that the method known as SHAP Values serves as the cornerstone for this campaign. We will also see how this merge gives birth to a proposal for a canonical solution for XML under the name of the SHAP Framework which, lastly, we will apply to a case study from the Credit Risk Models domain while describing the current situation of ML in this industry.

To improve the narrative follow-up, we decided to append the proofs to all the presented results in the annexes along with a graph analysis for every feature from the practical case and the implementations for the used techniques. For validation or replication of the results, interested readers may check the GitHub Repository for this project.

https://github.com/Aleix-Sancho/Explainable-Machine-Learning

Finally and most importantly, I would like to thank to my supervisor, Alejandra Cabaña, for introducing me to XML and the trust and freedom for creativity during the development of this project. Also, to Miriam Amores and Lluis Vidal from Banco Sabadell for letting me join their team, learn the fundamentals of the sector that now I am able to introduce here and the support and guidance in my professional and personal decisions.

Contents

1. Introduction to Explainability

As everything comes at a cost, the empirical success of more complex ML algorithms brings the trade-off of interpretability for predictive power. First, we shall refer to **interpretability** as the passive characteristic of a model addressed to make itself understandable by a human in its decision-making process. As more of these models arise, it is necessary to make the distinction between transparent and black-box models.

1.1. Transparency and Black-box-ness

We label a model as **transparent** if it is understandable by itself in the sense of comprehending how the input information is processed and how the decision-making process is made from it. Within transparent models we encounter linear regressions, decision trees or k-Nearest Neighbors, among others. In contrast, a **black-box** model is characterized by its complexity. Due to the sophistication in the newer models aimed at achieving efficiency or the rise in the number of parameters, it is non-trivial to assess how the predictions are made from the inputs. Among black-box models we find Tree Ensembles like Random Forests, Support Vector Machines or Neural Networks.

Interpretability, as defined before, characterizes transparent models. This is a deficiency in black-box-ness that creates challenges while modelling such as blindness or the slow adoption of these technologies in certain sectors.

The direct consequence of non-interpretability among black-box models comes with the form of blindness. Interpretability is crucial in some modelling steps – like detecting and correcting biases in the decision-making process – and black-box-ness difficulties or even denies these challenges to the model. On the other hand, an indirect effect of non-interpretability appears as a slow adoption in non-IA-expert sectors like the medical, banking or judicial. Not only more complex techniques require more qualified employees but they may also create an aversion in them. The common denominator among these sectors is the demand for non-quantitative arguments – that might be also backed by quantitative analyses – in the decision making process which black-box-ness cannot satisfy.

In section 5.1 we will study the particular case of the financial sector to understand its current situation between these new technologies from both practical and regulatory perspectives.

1.2. Explainability as a solution to black-box-ness

Given the problems that black-box models have to deal with, **Explainable ML** presents itself as a solution offering techniques to bring light to this opaqueness. In particular, it searches for building trust, confidence and fairness in these models or accessibility for users among others [1].

Explainable techniques act as post-hoc methods, in the sense that they are applied after the modeling process, and use certain means to enhance interpretability. In this project we will consider only local and feature relevance explanations as examples of them.

Local explanations techniques aim to segment the solution space and give the proper explanations in these less complex regions. This is usually achieved by using differentiation and Taylor Series properties. In this project we will study LIME, which is among the most popular techniques in this category. On the other hand, **feature relevance methods** seek explanations through computing relevance scores for each feature that quantify their importance in the decision-making. In this category we will study DeepLIFT and SHAP Values.

While designing these methods, we encounter two well-distinguishable paradigms: model-agnostic and model-specific techniques. **Model-agnostic techniques** are made to be applied to any kind of model regardless of their inner structure. This allows them to be reusable for new advancements. However, this creates a lack of efficiency since they cannot take any advantages at all on how the models function internally. In this type we position LIME and SHAP Values. Their opposite are **Model-specific techniques** which leverage on the model's internal characteristics. With this property these methods are able to achieve efficiency while explaining opaque models. Nevertheless, their value as methods cannot be transferred to other models to which they are not designed for. Here we find DeepLIFT which searches to take an advantage of neural network's inner structure.

This project has as purpose the study and exploration of how these techniques – LIME, DeepLIFT and SHAP Values – work and how they may be unified in the known SHAP Framework, which makes it a preferred candidate to canonize a solution for Explainable ML.

2. LIME as an example of a model-agnostic technique

In this section we will study how **Local Interpretable Model-Agnostic Explanations (LIME)** works as a model-agnostic and local explanations method and how we may interpret some of its solutions. In particular, given a point from the input space, LIME searches an explainable model that locally explains the model function around it – we may refer to **model function** as the function retrieved by the model after it has been trained –. First, we will introduce simplified inputs which is a frequent concept in Explainable ML and, finally, we shall study the geometric interpretation of its linear solutions as an approximation to the tangent plane of the model function at the selected input point.

Subsections 2.1 and 2.2 have been adapted from [2]. Subsection 2.3 has been supported by [3].

2.1. Simplified Inputs

With the aim of enhancing interpretability, **simplified inputs** are transformations of the features that change their interpretation. Formally, let $x \in \mathbb{R}^d$ be an observation from the input space. Then, its simplified input $x' \in \mathbb{R}^{d'}$ gets mapped to the original through the function h_x such that $x = h_x(x')$. With this definition, the mapping function depends on the original observation.

The choice for the simplified inputs relies heavily on the problem at hand. Consider x as a sequence of words, then x' may represent the presence or absence of certain words – observe how a single feature can be mapped to more than one simplified input –. Another example is to consider x as a real number interpreted as the age of a client in credit risk allowance and x' as a discrete version of it which classifies each individual as young, adult or senior.

In this project, we will focus on two general interpretations for simplified inputs. First, x' as the representation of the presence or absence of the corresponding feature in the model and, secondly, $x' = x$ – or $h_x = id$ – the original feature. In both cases, $d = d'$.

2.2. Constructing the explainable model

As mentioned before, LIME finds an explainable model to interpret the model function around a given point from the input space in a less complex manner. We shall now see how this simpler model is constructed.

Let $f : \mathbb{R}^d \to \mathbb{R}$ be the model function and $x \in \mathbb{R}^d$ the selected point for the local explanation. We shall now define G as the considered set of explainable models whose inputs will be the simplified inputs. Some examples of this set are G as the set of linear models, the set of decision tree models or their union. Therefore, the final explainable model will be a function of the form $g : \mathbb{R}^{d'} \to \mathbb{R} \in G$.

The explainable model is defined as the function solution of the following equation:

$$g = \min_{h \in G} \mathcal{L}(f, h, \pi_x) + \Omega(h)$$

where $\mathcal{L}$ is known as the fidelity function, π_x as the proximity measure and Ω as the complexity measure. In order for LIME to be an agnostic-model, all of these hyperparameters – parameters chosen by the programmer and not subjected to the model's training – depend exclusively on the problem at hand.

The **proximity measure** is a function of the form $\pi_x : \mathbb{R}^d \to \mathbb{R}$. Let $z \in \mathbb{R}^d$, then $\pi_x(z)$ quantifies the proximity of z to x. As an example consider π_x as the Gaussian kernel with hyperparameter the width σ.

$$\pi_x(z) = \exp\left(\frac{-||x - z||^2}{\sigma^2}\right)$$

This kernel equals 1 when $x = z$ and drops exponentially to 0 when z moves away from x.

The **fidelity function** $\mathcal{L}$ measures how faithful h is in approximating f in the locality defined by π_x. Therefore, $\mathcal{L}$ – with the help of π_x – guarantees the local fidelity for the explainable model. As an example of $\mathcal{L}$ we may consider the weighted sum of squares

$$\mathcal{L}(f, h, \pi_x) = \sum_{z, z' \in Z} \pi_x(z) \, \left(f(z) - h(z')\right)^2$$

where Z is a sample from the input space around x.

Finally, the **complexity measure** Ω enhances interpretability quantifying h's complexity. If h is a linear model we may consider it as the number of non-zero parameters – which may be implemented as a Lasso regression –. Or, if h is a decision tree, we could think of it as the tree's depth.

With these three hyperparameters, g becomes as both an explicable and locally faithful model and allows to interpret the model function in local regions.

2.3. A geometric interpretation for the explainable model

As said in the introduction to this section, when choosing the explainable model as a linear function, we may interpret it as an approximation to the tangent plane of the model function at the selected point from the input space. We will now study this geometrical interpretation of LIME's solutions and some of their peculiarities. For this purpose, we will consider from now on $h_x = id$. Formally, this result is reflected in the following lemma.

Lemma 2.1. *Let $f : \mathbb{R}^d \to \mathbb{R}$ be the model function, $x \in \mathbb{R}^d$ the selected point from the input space and $g(z) = \beta_0 + \sum_{i=1}^d \beta_i z_i$ be the linear explainable function with intercept given by LIME when $\mathcal{L}$ is the weighted linear regression, $\pi_x = 1$ and $\Omega = 0$. Consider the sample around x, $Y = \{y_i = f(z_i) | z_i = x + \delta_i\}_{i=1}^n$, $\delta_i \sim N(0, \sigma^2 I_n)$. Then, the estimated coefficients for g verify the equation*

$$\hat{\beta} = \begin{pmatrix} f(x) - \sum_{i=1}^d \partial_i f(x) \cdot x_i \\ \nabla f(x) \end{pmatrix} + \sigma^2 C \, O(||\frac{\delta}{\sigma}||^2) \, \mathbf{1}$$

where C is a constant and $\mathbf{1}$ the vector of ones. Thus, the function

$$g(z) \approx f(x) + \sum_{i=1}^d \partial_i f(x_i) \cdot (z_i - x_i)$$

is an approximation of the tangent plane of f at x when σ^2 is small.

In this result – whose proof is in the annexes as every other result –, we substituted the proximity measure π_x with a local sample over x. This lemma shows how in absence of an analytical expression of f the function given by the model – or at least a function no one would try to differentiate –, LIME is able to estimate its partial derivatives evaluated in x through a weighted linear regression and, consequently, approximate its tangent plane.

The previous result also indicates the relevance on the used locality of x. The error term of $\hat{\beta}$ depends directly on the size of this locality and on f's curvature. Therefore, a well chosen kernel width becomes crucial for any particular f.

As an example, we considered the following setup. Let $y = \sin(x)$ be the objective function. Then, we trained a dense neural network with two hidden layers, each with 64 neurons and RELU as the activation function, and an output layer with a single neuron and a linear activation function over a sample $\sin(x) + \epsilon$, $\epsilon \sim N(0, 0.5)$. The model function obtained along with the objective function and the used sample are displayed in Figure 1.

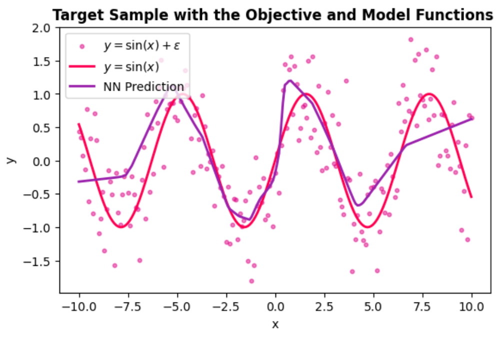


FIG. 1. Target Sample with the Objective and Model Functions.

Implementing LIME's solution as a linear regression over the feature x and around the point $x = 0$ – specifically with a sample in $(-.2, .2)$ –, we obtained the results shown in Figure 2 which also displays the tangent line of $y = \sin(x)$ at $x = 0$ that LIME is trying to approximate.

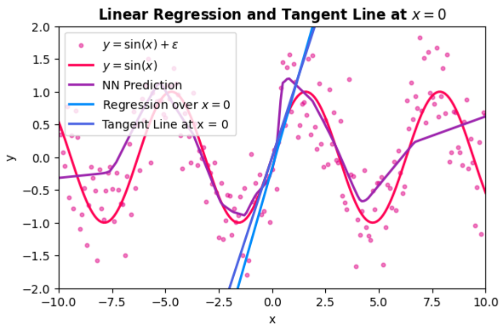


FIG. 2. Linear Regression with a Sample over $(-.2, .2)$ and Tangent Line at $x = 0$.

As shown, LIME's solution does a great job approximating this tangent plane – a line with the dimensionality in these function spaces –. However, the sample radius is crucial in this fitting, which is equivalent to choosing the right width for the kernel. Indeed, Figure 3 shows this estimation when the sample is over the interval $(-5, 5)$.

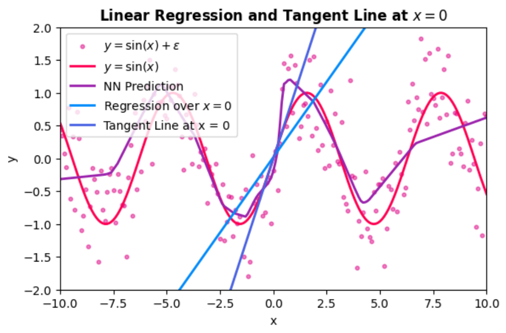


FIG. 3. Linear Regression with a Sample over $(-5, 5)$ and Tangent Line at $x = 0$.

Here, clearly the fitting is poorer showing the relevance of the kernel width. Dispensing with the kernel is neither an option as over-concentrated samples around the selected input point may cause instability in the results [3]. But, as shown, with carefully made selections, LIME becomes a great tool to locally interpret black-box models.

3. DeepLIFT as an example of a model-specific technique

We will now explore how **Deep Learning Important FeaTures (DeepLIFT)** deals with black-box-ness. As a model-specific technique, it leverages neural networks' inner structure to achieve efficiency. In particular, DeepLIFT takes advantage of the backpropagation component of neural networks.

Given a reference and a selected input, DeepLIFT explains the difference between the twp outtputs in terms of the difference between their inputs and assigns a contribution score to each feature, which assesses their contribution in the prediction. Therefore, DeepLIFT is classified mostly as a feature relevance method.

In this section, we will first analyse which properties are desired for contribution scores to have in terms of both the inputs and to allow the backpropagation of these importance signals from the output to the input layer. Then, we will see rules for choosing these contribution scores given their properties by taking a look to a single neuron. Finally, we will explore how applying these differences from reference values is equivalent to linearize the non-linear parts of the network and its advantages over other approaches.

This whole section has been adapted from the original paper [4].

3.1. Contribution Scores Properties

Let z represent the output of some target neuron and consider $x_1, ..., x_n$ some neurons' output from an intermediate layer such that they are sufficient to compute z. Also consider $x_1^0, ..., x_n^0$ and z^0 as the reference inputs and their prediction in the target neuron, respectively. Now we shall define $\{\Delta x_i = x_i - x_i^0\}_{i=1}^n$ and $\Delta z = z - z^0$ as their respective **difference-from-references** and denote $C_{\Delta x_i \Delta z}$ as the **contribution score** of Δx_i in Δz which aims to quantify the importance of x_i in predicting z.

Given DeepLIFT's locality - in choosing the values $x_1, ..., x_n$ -, a first desirable property for contribution scores is the following:

Property 3.1 (**Summation to Delta**). *The contribution scores of Δx_i to Δz, $\{ C_{\Delta x_i \Delta z}\}_{i=1}^n$, satisfy the equation*

$$\sum_{i=1}^{n} C_{\Delta x_i \Delta z} = \Delta z$$

Therefore, $C_{\Delta x_i \Delta z}$ may be interpreted as the amount of difference-from-reference in z attributed on the difference-from-reference in x_i.

Now, as a feature relevance technique, we expect to calculate the contribution scores from each feature x_i to the model's output f, $C_{\Delta x_i \Delta f}$. In order to do so, DeepLIFT uses a backpropagation-like algorithm where it calculates the contribution scores for each neuron in the hidden layers to the output layer and propagate them backwards until the input layer. These type of methods are known as Backpropagation-Based Approaches.

To see how this is achieved, first we will need the following definition. Let x be an input neuron and y some target neuron. Then, the **multiplier of** x **to** z denoted as $m_{\Delta x_i \Delta z}$ is defined as

$$m_{\Delta x \Delta z} = \frac{C_{\Delta x \Delta z}}{\Delta x}$$

To allow backpropagation we require from multipliers the next property:

Property 3.2 (**Chain Rule for Multipliers**). *Consider an intermediate layer with neurons $x_1, ..., x_n$, a target neuron z and a layer between them with neurons $y_1, ..., y_m$. Then, given the multipliers $\{m_{\Delta x_i \Delta y_j}\}_{i=1}^n$ and $\{m_{\Delta y_j \Delta z}\}_{j=1}^m$, the multipliers of x_i to z satisfy the following equation:*

$$m_{\Delta x_i \Delta z} = \sum_{i=1}^{m} m_{\Delta x_i \Delta y_j}\, m_{\Delta y_j \Delta z}$$

Both the Summation to Delta and the Chain Rule for Multipliers properties are compatible in the sense of the following lemma.

Lemma 3.1. *Consider $\{C_{\Delta x_i \Delta y_j}\}_{i=1,j=1}^{n,m}$ and $\{C_{\Delta y_j \Delta z}\}_{j=1}^m$ both satisfying the Summation to Delta property and $\{m_{\Delta x_i \Delta z}\}_{i=1}^n$ verifying the Chain Rule for Multipliers. Then $\{C_{\Delta x_i \Delta z}\}_{i=1}^n$ follows the Summation to Delta Property.*

Thus, given the multipliers for a layer's neurons, we can compute the contribution scores for any neuron in a previous layer efficiently via backpropagation until we arrive to the input layer containing the features. This algorithm is represented in Figure 4 for dense neural networks. Curious readers may refer to the annexes to check the implementation of DeepLIFT from this project for further details.

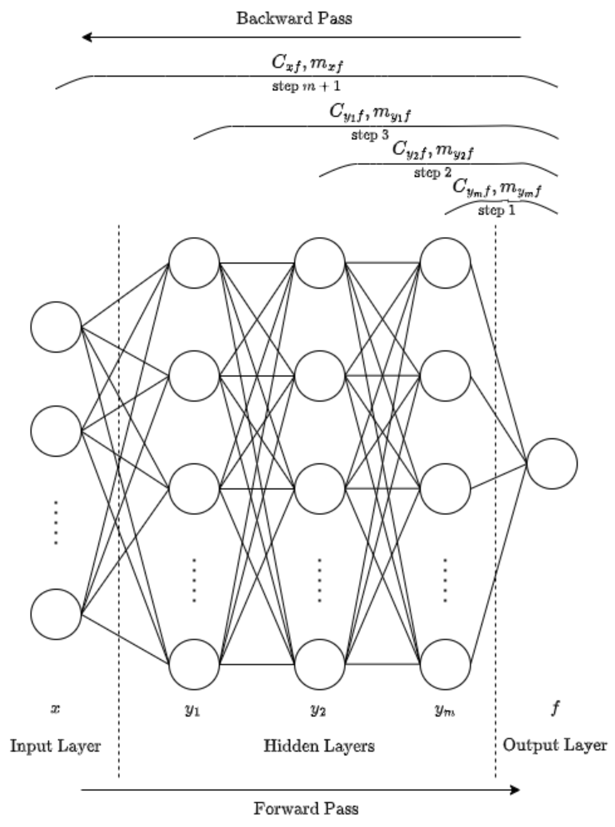


FIG. 4. Contribution Scores Calculation Algorithm for Dense Neural Networks Diagram.

3.2. Rules for Assigning Contribution Scores

Contribution Scores, defined until this point, are a set of solutions to the feature relevance problem that satisfy certain properties. However, we shall now introduce some rules to assign them specific values. To do so, consider first the general neuron from Figure 5 where we may think of $\{x_i\}_{i=1}^n$ as the features, $y = \omega_0 + \sum_{i=1}^n \omega_i\, x_i$ is the neuron's linear component and $z = r(y)$ is the non-linear component where r is the activation function. Also, let $\{x_i^0\}_{i=1}^n$, y^0 and $z^0 := r(y^0)$ be the reference values.

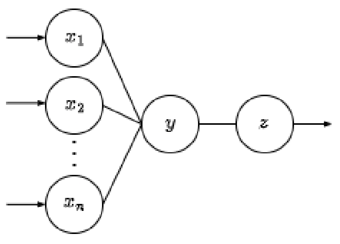


FIG. 5. General Neuron Diagram.

Starting with the non-linear part, the Summation to Delta property – StD from now on – states that $C_{\Delta y \Delta z} = \Delta z$ as r is a single-variable function. Therefore, $m_{\Delta y \Delta z} = \frac{\Delta z}{\Delta y}$. These assignments are grouped under the **Reescale Rule**[1].

Now, for the linear component, observe that $\Delta y = \sum_{i=1}^{n} \omega_i \Delta x_i$. Therefore, in order to be compatible with the StD property, a good candidate for the contribution scores is $C_{\Delta x_i \Delta y} = \omega_i \Delta x_i$ and, consequently, their multipliers are $m_{\Delta x_i \Delta y} = \omega_i$. This rule is known as the **Linear Rule**.

As an example of calculation, consider a neuron like the previous one with two inputs such that $x = (3, 4)$, $x^0 = (0, 0)$, $w = (-1, 0.5, 1.5)$ and $r = \text{RELU}$. First, we must calculate for each neuron's component their output for both inputs. After a quick computation we get $y = z = 6.5$, $y^0 = -1$ and $z^0 = 0$. This step is known as the **forward pass**. Now, we compute the contribution scores and multipliers for each neuron to its immediate successor. By the Reescale Rule $C_{\Delta y \Delta z} = \Delta z$ and $m_{\Delta y \Delta z} = \frac{\Delta z}{\Delta y}$. Also, by the Linear Rule we obtain $C_{\Delta x_i \Delta y} = \omega_i \Delta x_i$ and $m_{\Delta x_i \Delta y} = \omega_i$. Finally, applying the Chain Rule for Multipliers – CRfM from now on – we achieve the contributions scores from the inputs to the outputs of the neuron. Indeed:

$$\begin{aligned} C_{\Delta x_i \Delta z} &= \Delta x_i \, m_{\Delta x_i \Delta z} = \Delta x_i \, m_{\Delta x_i \Delta y} \, m_{\Delta y \Delta z} \\ &= C_{\Delta x_i \Delta y} \, m_{\Delta y \Delta z} = \omega_i \, \Delta x_i \, \frac{\Delta z}{\Delta y} \end{aligned}$$

and $C_{\Delta x_1 \Delta z} = 1.3$ and $C_{\Delta x_2 \Delta z} = 5.2$. Observe how, as both the StD and CRfM properties are compatible, $\{C_{\Delta x_i \Delta z}\}_{i=1}^{2}$ verifies the StD property such that $C_{\Delta x_1 \Delta z} + C_{\Delta x_2 \Delta z} = \Delta z = 6.5$. The backpropagation from the output to the input layer is known as the **backward pass**.

In a deeper network, using the CRfM property, we would transfer the contribution scores for each layer backwards until the input layer as shown in Figure 4. This role of transferring the scores via the multipliers takes the form in this example as the relationship $C_{\Delta x_i \Delta z} = C_{\Delta x_i \Delta y} \, m_{\Delta y \Delta z}$ observed in the previous calculation.

[1]In DeepLIFT's original paper its authors introduce now the concepts of both the positive and negative components for each difference-from-reference. From an introductory perspective, the presented definitions here are sufficient.

3.3. Differences-from-reference as linearizing the network's non-linear components

To end this section, we will study how using the difference-from-reference paradigm – with both the Linear and Reescale rules – is equivalent to linearize the non-linear parts of the network. Furthermore, we will question this framework by comparing it to the variant known as Gradient-Base methods.

Consider the general neuron from the previous subsection. By the Linear Rule we obtain that the multiplier from x_i to y equals y's partial derivative $m_{\Delta x_i \Delta y} = \omega_i = \partial_{x_i} y$. Therefore, applying the Reescale Rule, which considers $m_{\Delta y \Delta z} = \frac{\Delta z}{\Delta y}$, is equivalent to linearize the neuron's non-linear component. Indeed, it is the same as considering a differentiable function f that satisfies the equation $f(x) = f(x_0) + f'(x)\,(x - x_0)$ for every $x \in \mathbb{R}$. Then $f''(x)\,(x - x_0) = 0$ and $f'' = 0$. Thus, due to its Taylor Series, f is linear.

In consequence, we may interpret the multiplier of x to y as the x linearized partial derivative of y, which we will denote by $m_{\Delta x \Delta y} = \partial_x^* y$. The contribution score $C_{\Delta x \Delta y}$ is, then, this linearized partial derivative $\partial_x^* y$ weighted by Δx. With this approach, the CRfM property becomes in just the chain rule from calculus applied to this linear functions and centering y in the reference input we derive the StD property:

$$y = y^0 + \sum_i \partial_{x_i}^* y \,(x_i - x_i^0)$$

$$\Delta y = \sum_i \partial_{x_i}^* y \,\Delta x_i = \sum_i m_{\Delta x_i \Delta y} \Delta x_i = \sum_i C_{\Delta x \Delta y}$$

On the other hand, consider the non-linear function r such that $z = r(y)$. By the Mean Value Theorem we know that

$$m_{\Delta y \Delta z} = \frac{\Delta z}{\Delta y} = \frac{\Delta r}{\Delta y} = r'(c)$$

for some $c \in (y, y^0)$. With this in mind it is natural to ask why not to choose as a multiplier simply $r'(y)$ or $r'(y^0)$. And, in that case, why even bother choosing a reference input. These kind of methods where the gradient is used as the importance signal are known as Gradient-Based Methods. To answer these questions we will focus on RELU as the activation function, study some disadvantages of using the gradient and how differences-from-reference approach them through the examples offered by DeepLIFT's authors.

Consider the simple network from Figure 6

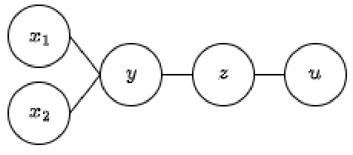


FIG. 6. Simple Network Diagram.

where $y = 1 - x_1 - x_2$, $z = \text{RELU}(y)$ and $u = 1 - z$. Let the inputs be $x = (1, 1)$ and $x^0 = (0, 0)$. With a quick computation we get the network's output as:

$$u = \begin{cases} x_1 + x_2, & x_1 + x_2 < 1 \\ 1, & x_1 + x_2 \geq 1 \end{cases}$$

and the selected outputs for each neuron are $(y, y^0) = (-1, 1)$, $(z, z^0) = (0, 1)$ and $(u, u^0) = (1, 0)$. In this particular scenario, observe how slight changes in both x_1 and x_2 do not alter the network's output. This type of situations are known as **model saturation**.

A gradient-based approach would use u's gradient evaluated at x to propagate importance signals – evaluating at the reference would give us a constant importance value for any target input –. However, the gradient fails in model saturation situations as both $\partial_{x_1} u(1) = \partial_{x_2} u(1) = \partial_{x_i}(1) = 0$ when $x_1 + x_2 \geq 1$, which underestimates both importance signals as the network's output is non-zero.

Using differences-from-reference, contribution scores do not fail in model saturation scenarios. Indeed, reusing the calculations from the previous section's example, we obtain:

$$C_{\Delta x_i \Delta u} = C_{\Delta x_i \Delta z} \, m_{\Delta z \Delta u} = \frac{\Delta x_i}{2}$$

Another gradient-based approaches' undesirable property is the **discontinuity of importance signals**. RELU, as the most popular activation function, causes a discontinuity in the following partial derivative

$$\partial_{x_i} z = \begin{cases} -1, & x_1 + x_2 < 1 \\ 0, & x_1 + x_2 \geq 1 \end{cases}$$

that influences to these importance signals. By contrast, the contribution scores $C_{\Delta x_i \Delta z} = \omega_i \, \Delta x_i \, \frac{\Delta z}{\Delta y}$ with the reference inputs fixed are continuous.

Aggregating both cases – model saturation and importance signal discontinuities –, differences-from-reference become a preferred solution over gradient-based approaches for transferring these contribution signals.

4. Cooperative Games and Explainability

As seen before, feature relevance approaches aim to solve the explainability problem of assigning contribution scores given a set of features for a specific model's prediction. We now shall see how this paradigm may be reframed as a cooperative game which will allow us to leverage Game Theory results and link it with local explanations techniques.

We will start by introducing the required Cooperative Games fundamentals and their preferred solution: the Shapley Values. Then, we will reframe the Feature Relevance Problem – FRP from now on – and its main variant: the SHAP Values, which are the central part of this project. They not only provide a solution to the FRP based on robust mathematical results but they are also the variant of Shapley Values that allows establishing a connection between them and both LIME and DeepLIFT. This unique condition positions them as the first candidate to canonize a solution for the explainability problem under the name of the SHAP Framework.

The Cooperative Games Theory fundamentals in subsections 4.1 and 4.2 are extracted from [5]. Their link with ML and the SHAP Framework has been adapted from [6].

4.1. Cooperative Games Theory

A **cooperative game** is a game where a set of players form a coalition to achieve a common objective that grants a reward. The problem at hand is how to allocate the payoff among these players given their contributions to the coalition. To ease the modelling, cooperative games introduce the concept of **transferable utility** which may be interpreted as a commodity (or money), a payment method free to be exchanged that is received after the cooperation.

Formally, let $\mathcal{N}$ be a finite set with n players, which is known as the **grand coalition**, and $\nu : 2^{\mathcal{N}} \to \mathbb{R}$ be a characteristic function such that $2^{\mathcal{N}}$ is the set of every possible $\mathcal{N}$'s subset and $\nu(\emptyset) = 0$. Given a (sub)coalition $S \subseteq \mathcal{N}$, $\nu(S)$ represents the worth of S which is the amount of transferable unity that the players in S would receive without the help of any member out of S – observe how, coherently, the empty coalition's worth is 0 –. Thus, we define a cooperative game as the pair $(\mathcal{N}, \nu)$.

Consider as an example the grand coalition as a set with three players $\mathcal{N} = \{1, 2, 3\}$ and the characteristic function ν such that:

$$\nu(\{1\}) = 10 \quad \nu(\{2\}) = 15 \quad \nu(\{3\}) = 0$$
$$\nu(\{1,2\}) = 30 \quad \nu(\{1,3\}) = 10 \quad \nu(\{2,3\}) = 15$$
$$\nu(\{\emptyset\}) = 0 \quad \nu(\{1,2,3\}) = 30$$

Here $v(\{1, 2\}) = 30$ represents the worth – or the amount of transferable utility – players 1 and 2 achieve without any help from player 3. And idem for every other subset S.

The solution to a cooperative game is given by the function $\phi : \mathbb{R}^{\mathcal{L}(\mathcal{N})} \to \mathbb{R}^n$ where $\mathbb{R}^{\mathcal{L}(\mathcal{N})}$ is the set of every characteristic function defined over $\mathcal{N}$. The real-valued vector $\phi(\nu) = (\phi_i(\nu))_{i \in \mathcal{N}}$, known as the **payoff vector**, retrieves the amount of transferable utility $\phi_i(\nu)$ that player i receives for participating in the grand coalition. When ν is implicitly known, we will denote $\phi_i := \phi_i(\nu)$ and $\phi := \phi(\nu)$.

From the previous example, consider the payoff vector $\phi(\nu) = (10, 10, 10)$ which is an equal distribution of $\nu(\mathcal{N})$. However, taking a look to ν, equality does not seem fair when the contribution of player 3 is non-existent. This fact creates the demand to expect certain properties from the payoff vector. This idea reaches until the Shapley Values solution which we now shall see.

4.2. Shapley Values

Shapley Values deal with the payoff vector search by finding a fair solution, in the sense that they establish a set of desirable properties, that characterizes them as a unique solution. Now, we will state these three necessary and sufficient properties to prove their uniqueness.

Before enunciating the first property, consider $\pi : \mathcal{N} \to \mathcal{N}$ a permutation and $\nu \in \mathbb{R}^{\mathcal{L}(\mathcal{N})}$ a characteristic function. Then we define $\pi\nu \in \mathbb{R}^{\mathcal{L}(\mathcal{N})}$ as the characteristic function that satisfies the equation

$$\pi\nu\left(\{\pi(i) : i \in S\}\right) = \nu(S)$$

for every $S \subseteq \mathcal{N}$. This represents that the role of player i in ν is the same as the role of $\pi(i)$ in $\pi\nu$.

Property 4.1 (**Symmetry**). *Let $\nu \in \mathbb{R}^{\mathcal{L}(\mathcal{N})}$ and $\pi : \mathcal{N} \to \mathcal{N}$ be a permutation. Then, for every player $i \in \mathcal{N}$*

$$\phi_{\pi(i)}(\pi\nu) = \phi_i(\nu)$$

Symmetry establishes the desired property that the payoff for each player only depends on its contribution to the coalition and not on their role.

For the next property we need the following definition. Let $R \subseteq \mathcal{N}$. Then, R is a **carrier** of ν if

$$\nu(S \cap R) = \nu(S)$$

for every $S \subseteq \mathcal{N}$. In particular, $\nu(R) = \nu(\mathcal{N} \cap R) = \nu(\mathcal{N})$. Therefore, all the grand coalition's effort is allocated among the players in R. For every player outside R its worth is 0. Indeed, let $i \notin R$ then $\nu(\{i\}) = \nu(\{i\} \cap R) = \nu(\emptyset) = 0$. This kind of players are known as **dummies** as they do not contribute in the coalition at all.

Property 4.2 (**Carrier**). *Let $\nu \in \mathbb{R}^{\mathcal{L}(\mathcal{N})}$ and R be a carrier of ν. Then*

$$\sum_{i \in R} \phi_i(\nu) = \nu(R)$$

The Carrier Property establishes that the payoff is distributed only among players in a carrier since, as stated before, $\nu(R) = \nu(\mathcal{N})$.

Observe from the example in the previous section, $R = \{1, 2\}$ is a carrier of ν. However, the equal distribution $\phi = (10, 10, 10)$ does not satisfy the Carrier Property. Indeed, player 3 is a dummy player and $\phi_1 + \phi_2 \neq \nu(\mathcal{N})$. Thus, following the Shapley Properties, it is not a desirable solution.

For the last property, let $a, b \in \mathbb{R}$ and $\nu, \omega \in \mathbb{R}^{\mathcal{L}(\mathcal{N})}$. Then, we define $a\nu + b\omega \in \mathbb{R}^{\mathcal{L}(\mathcal{N})}$ as the characteristic function that satisfies the equation

$$(a\nu + b\omega)(S) = a \cdot \nu(S) + b \cdot \nu(S)$$

for every $S \subseteq \mathcal{N}$.

Property 4.3 (**Linearity**). *Let $a, b \in \mathbb{R}$ and $\nu, \omega \in \mathbb{R}^{\mathcal{L}(\mathcal{N})}$. Then,*

$$\phi(a\nu + b\omega) = a \cdot \phi(\nu) + b \cdot \phi(\omega)$$

To interpret the Linear Property we may consider both additivity and homogeneity separately. For additivity, the payoff vector in the cooperative game $(\mathcal{N}, \nu + \omega)$ represents trivially the sum of the payoffs from the games $(\mathcal{N}, \nu)$ and $(\mathcal{N}, \omega)$. For homogeneity, consider the game $(\mathcal{N}, a \cdot \nu)$. Then, by the Linear Property, $\phi_i(a\nu) = a \cdot \phi_i(\nu)$. This payoff transformation may be interpreted as, if the reward for the grand coalition changes by a factor a, the allocation for each player should also be changed by the same factor.

With these three properties now we are able to proof Shapley Values' uniqueness.

Theorem 4.1 (**Shapley Values**). *Let $(\mathcal{N}, \nu)$ be a cooperative game. Then there exists a unique solution $\phi : \mathbb{R}^{\mathcal{L}(\mathcal{N})} \to \mathbb{R}^n$ satisfying all Symmetry, Carrier and Linear Properties. This solution verifies the following equation*

$$\phi_i(\nu) = \sum_{S \subseteq \mathcal{N} \setminus \{i\}} \frac{|S|!\,(|\mathcal{N}| - |S| - 1)!}{|\mathcal{N}|!} \left(\nu(S \cup \{i\}) - \nu(S)\right)$$

and its values are known as Shapley Values.

The formula for Shapley Values may be interpreted as follows. Consider an arrange for the n players in a queue such that each one makes its contribution sequentially. Then, for each coalition $S \subseteq \mathcal{N}$ such that $i \notin S$ there are $|S|!\,(|N| - |S| - 1)!$ queue's permutations where the players in S participate in the coalition before i does. As every permutation is as likely to happen, the probability that every player in S is immediately before i in the queue is $|S|!\,(|N| - |S| - 1)!\,/\,|N|!$. Furthermore, its marginal contribution after joining the coalition is $\nu(S \cup \{i\}) - \nu(S)$.

Thus, the **Shapley Value** for a player equals the average of its marginal contributions in all possible coalitions from the grand coalition.

Consider the example from the previous subsection one last time. Then, the Shapley Values for this cooperative game are

$$\begin{array}{lcccccccccl}
\phi_1 & = & \frac{0!\,2!}{3!} \cdot 10 & + & \frac{1!\,1!}{3!} \cdot 15 & + & \frac{1!\,1!}{3!} \cdot 10 & + & \frac{2!\,0!}{3!} \cdot 15 & = & 12.5 \\
S: & & \emptyset & & \{2\} & & \{3\} & & \{2,3\} & & \\
\phi_2 & = & \frac{0!\,2!}{3!} \cdot 15 & + & \frac{1!\,1!}{3!} \cdot 20 & + & \frac{1!\,1!}{3!} \cdot 15 & + & \frac{2!\,0!}{3!} \cdot 20 & = & 17.5 \\
S: & & \emptyset & & \{1\} & & \{3\} & & \{1,3\} & & \\
\phi_3 & = & \frac{0!\,2!}{3!} \cdot 0 & + & \frac{1!\,1!}{3!} \cdot 0 & + & \frac{1!\,1!}{3!} \cdot 0 & + & \frac{2!\,0!}{3!} \cdot 0 & = & 0 \\
S: & & \emptyset & & \{1\} & & \{2\} & & \{1,2\} & &
\end{array}$$

Observe how, as $R = \{1, 2\}$ is a carrier, we obtain $\phi_1 + \phi_2 = \nu(\mathcal{N}) = 30$. Another property that Shapley Values guarantee is the **null-player** property where dummy players, like player 3, do not receive any payoff $\phi_3 = 0$. An additional property from this solution is **efficiency**, which states that $\sum_{i=1}^{n} \phi_i = \nu(\mathcal{N})$.

4.3. Additive Feature Attribution Methods

The main objective for feature relevance is to assign contribution scores for each feature in order to quantify their contribution in the chosen prediction. Therefore, it is plausible to treat the FRP as a cooperative game where each feature is represented as a player and the payoff to distribute is the predicted value. While doing so, we shall leverage on the results from previous sections to offer a solution to the FRP that is both fair – in the already stated sense – and mathematically robust.

To reframe the problem formally, consider F as the set of features, $d = |F|$ and $F' = \{0,1\}^d$ the set of simplified inputs where $z' \in F'$ represents the presence or absence of $z \in F$ in the model – see section 2.1 –. Then, we define the following.

Definition 4.1 (**Additive Feature Attribution Method**). *An Additive Feature Attribution Method is an explainability method with explainable model a linear function over the simplified outputs*

$$g(z') = \phi_0 + \sum_{i=0}^{d} \phi_i \, z'_i$$

such that $z' \in F'$ and $\phi_i \in \mathbb{R}$

It is immediate that both Linear LIME and DeepLIFT are Additive Feature Attribution Methods – AFAMs from now on –. Indeed, for Linear LIME, as the explainable model g is a linear function over the simplified features – see section 2.2 – then it is automatically an AFAM. For DeepLIFT, let f be the model function and consider the StD property

$$\sum_{i=1}^{d} C_{\Delta x_i f} = \Delta f$$

Defining $\phi_i = C_{\Delta x_i f}$ and $\phi_0 = f(x^0)$ – see section 3.1 – we also obtain an AFAM.

AFAMs aim to achieve local explanations around a given observation by defining a linear explainable model. However, by interpreting its coefficients as contribution scores for each feature we can leverage the Shapley Values Theorem to guarantee them some desirable properties. To do so, observe that the simplified features' interpretation allow us to model every possible subcoalition. Therefore, defining the characteristic function $f_x := f \circ h_x$ and given a target observation x, we are able to define the **AFAM cooperative game** (F', f_x) with reward to distribute $f(x) - \phi_0$.

Now we shall restate the Shapley Properties and their Theorem in AFAM's context.

Property 4.4 (**Symmetry for AFAM**). *Let $\pi : F' \to F'$ be a permutation over the set of simplified features. Then, for each feature $x'_i \in F'$*

$$\phi_{\pi(x'_i)}(\pi f_x) = \phi_{x'_i}(f_x)$$

Property 4.5 (**Carrier for AFAM**). *Let $R \subseteq F'$ be a carrier of f_x. Then*

$$\sum_{x'_i \in R} \phi_i = f(x) - \phi_0$$

Property 4.6 (**Linearity for AFAM**). *Let $a, b \in \mathbb{R}$ and f_x, g_x be two model functions. Then*

$$\phi(a f_x + b g_x) = a\phi(f_x) + b\phi(g_x)$$

Theorem 4.2 (**Shapley Values for AFAM**). *Let (F', f_x) be an AFAM cooperative game. Then exists a unique explainable model*

$$g(z') = \phi_0 + \sum_{i=0}^{d} \phi_i z'_i$$

satisfying all Symmetry, Carrier and Linear Properties for AFAM[2]. Its coefficients verify the following

$$\phi_0(f_x) = f_x(\mathbf{0})$$

$$\phi_i(f_x) = \sum_{z' \subseteq x' \setminus \{i\}} \frac{|z'|!\,(d - |z'| - 1)!}{d!} \left(f_x(z' \cup \{i\}) - f_x(z')\right)$$

where $z' \subseteq x'$ represents all vector z' where the non-zero entries are a subset of the non-zero entries of x', $|z'| = \sum_{i=1}^{d} z'_i$ is the number of non-zero entries of z', $z' \cup i$ is the vector such that $(z' \cup i)_j = z'_j$ if $i \neq j$ and $(z' \cup i)_j = 1$ if $i = j$, and its equivalent for $z' \setminus i$.

Therefore, this solutions is the linear approximation over the simplified features of the model function whose coefficients are their Shapley Values. The efficiency property, in the Explainable ML context, is known as **local accuracy** which states that $f(x) = g(x') = \phi_0 + \sum_{i=1}^{d} \phi_i x'_i$. The null-player property is referred to as **missingness**, which establishes that if $x'_i = 0$ then $\phi_i = 0$.

Thus, among all AFAMs, which aim to be local explanations, the unique explainable model that ensures the three Shapley Properties for AFAM is the Shapley Values for AFAM, which is a concept from the feature relevance paradigm. In other words, Shapley Values establish a bridge between local a feature relevance explanations.

[2]The properties used in [6] to derive the Shapley Values for AFAM Theorem are Local Accuracy, Missingness and Consistency – refer to references to know more about them –. As the theorem is the Shapley Values Theorem for a particular cooperative game and we have proved it with the properties here presented, both sets of properties are equivalent for this campaign.

4.4. SHAP Values, Implementing Shapley Values for ML models

Recall that characteristic functions quantify the worth a subcoalition $S \subseteq F$ has without the help of any player out of S. Therefore, there is a direct implementation for Shapley Values for AFAM. Consider as the characteristic function $f_x = f_S$ the model function trained only over the features in S. Thus, its Shapley Values are

$$\phi_0 = f_{\emptyset}(x_{\emptyset}) = \mathbb{E}\left(f(x)\right)$$

$$\phi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!\left(|F| - |S| - 1\right)!}{|F|!} \left(f_{S\cup\{i\}}(x_{S\cup\{i\}}) - f_S(x_S)\right)$$

where x_S represents the values of the features in S. Here, ϕ_0 equals the expected prediction value that we could approximate by the mean of all the predictions for observations in the sample. This straight approach is known as **Shapley Regression Values**.

In addition, a more precise interpretation for Shapley Values may be appropriate for this solution. Observe that $g(\mathbf{0}) = \phi_0$ represents a starting point for the explainable function when every feature is missing. By adding features to the model – $x'_i = 1$ for some i –, their contributions get reflected as deviations from this base value. Then, a positive Shapley Value indicates that the prediction value increases with the presence of its corresponding feature and idem for a negative one. Their absolute values show the magnitude of its importance, which allows to rank the features. After including all features, the explainable function successfully arrives to the value $f(x)$ from ϕ_0 thanks to the local accuracy property.

However, Shapley Regression Values come with a computational cost on the order of $2^{|F|}$. Indeed, we may think of a specific combination of the presence and absence of features as a binary vector of length $|F|$. And there are $2^{|F|}$ of them. Therefore, not only we have to make that many computations but also we have to retrain the model each of them.

Shapley Additive Explanation Values (SHAP Values) address this last problem by considering as characteristic function $f_x = \mathbb{E}\left(f|x_S\right)$ as an approximation of f_S. With it, we only would have to retrain the model once and then compute $2^{|F|}$ expected values which can be estimated as the mean over the predictions made using the input sample.

Consider f as a linear function. Then

$$\begin{aligned}\mathbb{E}\left(f(z)\,|\,x_S\right) &= \mathbb{E}\left(\beta_0 + \sum_{i=1}^{d} \beta_i z_i | x_S\right) \\ &= \beta_0 + \sum_{z_i \in S} \beta_i z_i + \sum_{z_j \notin S} \beta_j \mathbb{E}\left(z_j\right) \\ &= f\left([z_S, \mathbb{E}\left(z_{\overline{S}}\right)]\right)\end{aligned}$$

where $z_{\overline{S}}$ represents the features out of S. Therefore, the value for this characteristic function gets reduced to a single prediction. For this purpose, linear approximations for the model function are commonly used.

To assess the effectiveness of this approximation we considered the following setup. First, let the objective function be the following linear function.

$$y = \beta_1 x_1 + \beta_2 x_2 + \epsilon$$

where $\beta = (2, -.5)$, $x_i \sim N(0, 2)$ and $\epsilon \sim N(0, 1)$. Then, we trained two distinct model functions, a first function where x_1 was the only feature that we will name f_1 and, secondly, another function where we used both x_1 and x_2 as features named f_2. For both functions we used as model a dense neural network with a unique hidden layer with 10 neurons and RELU as the activation function; and an output layer with one neuron and a linear activation function. For the training part we used 100 epochs for both models and 100 observations in the training sample.

The comparison consists in evaluating how good is $f_2(x_1, \mathbb{E}(x_2))$ at approximating $f_1(x_1)$. Observe that, as both features are centered in 0, the expected values are $\mathbb{E}(x_i) = 0$. Both model functions along with the sample for x_1 are shown in Figure 7.

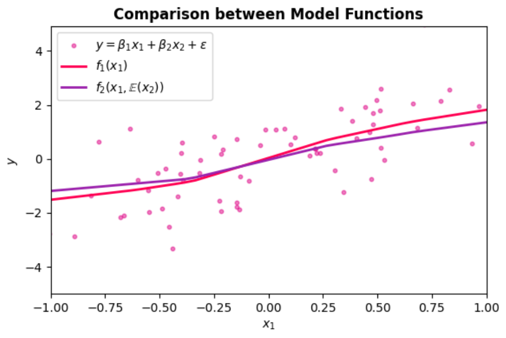


FIG. 7. Comparison between Model Functions.

As seen, f_2 with x_2 fixed at its mean does a great job approximating f_1. Also notice that using RELU may improve the results over other activation functions.

To further validate the robustness of this approach, we conducted additional experiments by varying the objective functions to include sine waves and quadratic functions based on the features. Through these experiments, we discovered that the fidelity of the approximation is highly dependent on the nature of the objective function. Specifically, while linear functions were well-approximated, sine waves presented a moderate challenge due to their periodic nature, and quadratic functions exhibited varying degrees of accuracy depending on the complexity of their coefficients. Nonetheless, in general, the method provided a fast and effective approximation to these diverse model functions using subsets of the features.

4.5. SHAP Values as a Unified Framework for Explainability

SHAP Values not only offer an efficient and theoretical robust approach to the FRP but also they enable a connection with the three methods seen until now – LIME, DeepLIFT and Shapley Regression Values –. As well pointed in [6], as both Linear LIME and DeepLIFT are AFAMs with eligible hyperparameters, it is naturally to ask if there is a specific choice of them such that it allows to recover the Shapley Values. Or equivalently, a choice that makes them verify all three Shapley Properties for AFAM. The answer is yes and it dissolves the arbitrarity in these hyperparameters.

For Linear LIME, recall from section 2.2 that its hyperparameters are $\mathcal{L}$, π_x and Ω. The choice that retrieves the Shapley Values is stated in the **Kernel SHAP** Theorem.

Theorem 4.3 (**Kernel SHAP**). *Consider the explainable model from Linear LIME*

$$g = \min_{h \in G} \mathcal{L}(f, h, \pi_x) + \Omega(h)$$

Then, the hyperparameters $\mathcal{L}$, π_x and Ω that retrieve the Shapley Values for AFAM are

$$\mathcal{L}(f, h, \pi_x) = \sum_{z' \in F'} \pi_{x'}(z') \, \left(f_x(z') - h(z')\right)^2$$

$$\pi_{x'}(z') = \frac{d-1}{\binom{d}{|z'|} |z'| (d - |z'|)}$$

$$\Omega(h) = 0$$

Observe that both $z' = \mathbf{0}$ and $z' = \mathbf{1}$ result in an infinite proximity measure value $\pi_{x'}(z') = \infty$. Therefore, $\phi_0 = f_x(\mathbf{0}) = \mathbb{E}\left(f(x)\right)$ and $\sum_{i=0}^{d} \phi_i = f_x(\mathbf{1}) = f(x)$. To solve this regression we may use these conditions to avoid infinite. Indeed, after a few algebraic manipulations, the equivalent model is:

$$f_x(z) + \left(\mathbb{E}(f(x)) - f(x)\right) z_1 - \mathbb{E}(f(x)) = \sum_{i=2}^{n} \phi_i (z_i - z_1)$$

Thus, a connection between Linear LIME and Shapley Values for AFAM is established. Observe how this theorem allows to use any characteristic function. Furthermore, this result shows us that Shapley Values for AFAM can be obtained through weighted linear regressions.

On the other hand, before introducing Deep SHAP Values, consider the following result.

Corollary 4.1 (**Linear SHAP**). *Let the model function be linear $f(z) = \beta_0 + \sum_{i=1}^{d} \beta_i z_i$. Then, the SHAP Values are*

$$\phi_0 = \beta_0$$

$$\phi_i = \beta_i \left(x_i - \mathbb{E}(z_i)\right)$$

DeepLIFT's unique hyperparameter is the reference input x^0. Instead of a theorem, **Deep SHAP** offers a fast approximation for SHAP Values when $x^0 = \mathbb{E}(x)$. Indeed, recall that references-from-reference is equivalent to linearize the network's non-linear components. Consider the neuron from Figure 5 from subsection 3.2. By the Linear Rule $C_{\Delta x_i \Delta y} = \omega_i \Delta x_i = \omega_i \left(x_i - \mathbb{E}(x_i)\right) = \phi_{x_i}(y)$ and by the Reescale Rule $C_{\Delta y \Delta z} = \Delta z / \Delta y = r'(c) \left(y - \mathbb{E}(y)\right) \approx \phi_y(z)$ for some $c \in (y, \mathbb{E}(y))$. Therefore by Corollary 4.1, for every neuron its inner SHAP Values are linearly approximated in the form of contribution scores. And, through backpropagation, these are converted in contribution scores for the whole network approximating the feature's SHAP Values. Thus, a bridge between Deep LIFT and SHAP Values is established.

Concluding, the whole picture for AFAMs is the shown in Figure 8.

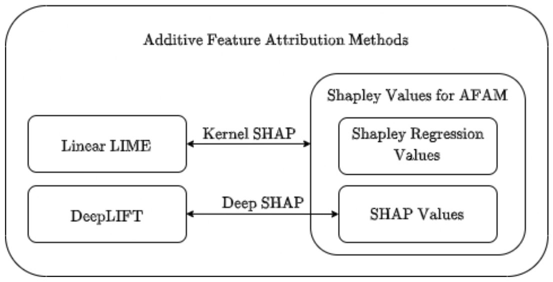


FIG. 8. AFAM's Relationship Diagram.

Observe how SHAP Values serve as the cornerstone for AFAMs and along with Kernel and Deep SHAP Values are not only a fair solution – see sections 4.2 –, but also they unify the explainable techniques seen – LIME, DeepLIFT and Shapley Values for AFAM –. Therefore, they become the first candidate to canonize a solution for Explainable ML under the name of the **SHAP Framework**.

To do a brief comparison between these three techniques consider as the objective function the linear function

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$$

where $\beta = (2, 10, -4)$, $x_i \sim N(0, 2)$ and $\epsilon \sim N(0, 1)$.

For the black-box model we considered a dense neural network with two hidden layers with 10 neurons each and RELU as the activation function; and an output layer with one neuron and without activation function. To train this model, we added a non-meaningful feature $x_3 \sim N(0, 2)$ and used the mean-squared-error function as the loss function. After training the model over 100 epochs, we obtained a 4.53 mean squared error on the train sample.

On the techniques implementations, while approaching this problem a first thought was to use the shap library from Python. However, we found an error in the SHAP Values retrieved using Tensorflow / Keras models. This error hap-

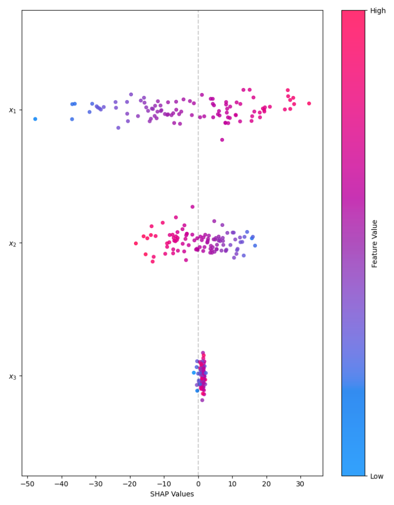


FIG. 9. SHAP Values for the comparison case.

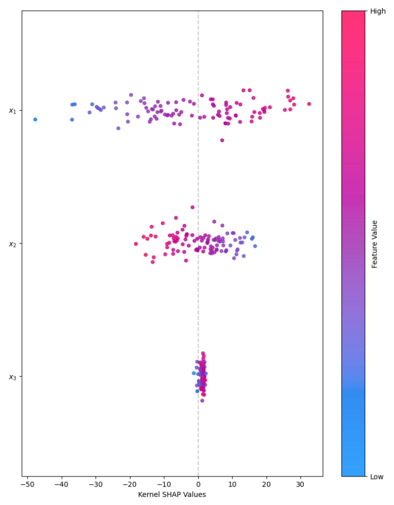


FIG. 10. Kernel SHAP Values for the comparison case.

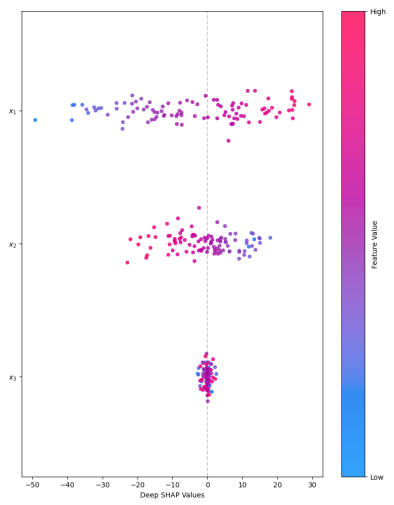


FIG. 11. Deep SHAP Values for the comparison case.

pened to be already spotted in forums[3] and without any solution by the time we encountered it. Thus, we decided to implement our own version of these techniques. Interested readers may find them in the annexes.

Then, we computed the SHAP Values, Figure 9; the Kernel SHAP Values, Figure 10; and the Deep SHAP Values, Figure 11. For all three techniques, we calculated the SHAP Values for a sample with 100 observations. Remember that here SHAP Values represent the deviation from the base value ϕ_0 attributed to the corresponding feature.

The first thing to notice is that both SHAP and Kernel SHAP Values are indistinguishable due to Theorem 4.3 which ensures that, in fact, they are equal. Also, as Deep SHAP Values are an approximation to SHAP Values their results are similar, but with subtle differences.

On the features, they are ranked by the higher SHAP value obtained in absolute value for each technique. The ranking proposed is the same for all three methods and, coherently, it matches with the ranking where the features are ordered by the absolute value of their corresponding coefficient in the objective linear function.

Feature x_1 not only ranks as the most important feature but also it shows an increasing correspondence with the target. Indeed, observe that for high values – pink dots – we obtain positive SHAP Values and for low values – blue dots – negative ones. This relation is aligned with the feature's coefficient $\beta_1 = 10$. For feature x_2, its coefficient is $\beta_2 = -4$. Therefore, it ranks after feature x_1 and it shows a non-increasing relation with the model's output. Lastly, feature x_3 does not affect the objective function value. Therefore, its SHAP Values are concentrated around 0 and it ranks as the least important feature.

On one hand, this alignment with the objective function shows that the neural network has been successfully adjusted to the objective function. But also, it manifests the competence of SHAP Values to identify both the relation with the output and the importance for each feature endorsing the SHAP Framework with empirical evidence – besides the theoretical foundation mentioned earlier –.

5. Applying the SHAP Framework to a Credit Risk Model

In this final section, we will study an application of the SHAP Framework to a credit risk dataset and explore its capabilities. This will include how this paradigm can provide insights into the factors from both financial intuition and quantitative perspectives. But first, we will see how this new Explainability Paradigm is reflected in the industry.

[3]https://github.com/shap/shap/issues/3612

5.1. Opportunities and Challenges for Explainability in Credit Risk

Credit risk is the effects or losses a financial entity assumes as a consequence of borrowers, or obligors, failing to meet their obligations to repay their obligations, which is known as a **default**. Credit risk management arises then as the set of strategies to identify, assess and mitigate it. Effective credit risk management involves, among others, the use of ML models to minimize potential losses.

Institutions, which most of them are banks, are subject to strict regulations from the competent authorities in order to protect depositors and bring confidence and stability to the financial system [7]. As a brief example, banks are required to keep a minimum amount of capital for the risks taken known as **regulatory capital** – even though the may save additional capital, known as economic capital, based on internal assessments –. In consequence, international authorities periodically – and especially after the 2008 financial crisis – update regulations to adapt it to the newest advancements. Examples of competent authorities are the Basel Committee on Banking Supervision or the European Banking Authority (EBA).

In our matter, newer ML models are yet unused from institutions in heavily regulated activities – such us the estimation of regulatory capital – in avoidance of conflict with supervisors. Both financial and regulatory entities are aware of the challenges and opportunities that arise with these new technologies. However, the precaution is mandatory [8].

Consider the **Capital Requirements Regulations (CRR)** issued by the European Parliament in 2013. As one may sense, it regulates the capital requirements for financial institutions in EU. Its article 174 [9] starts as follows.

Article 174
Use of models
If an institution uses statistical models and other mechanical methods to assign exposures to obligors or facilitates grades or pools, the following requirements shall be met.

Even though specific ML methods are not explicitly prohibited, there are certain demands and regulatory requirements imposed on the models used, which tend to favor the use of transparent models. One of these requirements is the following

(a) The model shall have good predictive power and capital requirements shall not be distorted as a result of its use. The input variables shall form a reasonable and effective basis for the resulting predictions. The model shall not have material biases.

As already mentioned in section 1.1, even though black-box models usually achieve a great predictive power, they obstruct some of the challenges to the modelling process such us bias detection. This is just one of the many examples where, in order to avoid conflicts with regulatory supervisors, financial entities would prefer using transparent instead of black-box models. Explainable techniques help to mitigate these difficulties – which would ease theoretically, but harder in practice, non-IA-experts – and some of them are already in use in certain financial areas – as the author has observed –. However, even though some of the path has already been traveled, there is still a distance to cover in order to fully exploit these new opportunities from ML and see the maximum potential in compatibility with the financial sector and beyond.

5.2. A Credit Risk Model

For the dataset, we considered a Credit Risk Dataset from Kaggle[4] that aims to predict the loans' status – whether it is a default or not – given some features.

The dataset contains the following features:

Age, of the borrower
Income, of the borrower
Employment Length, of the borrower
Loan Amount,
Interest Rate, applied to the loan
Percent Income, with respect to the loan amount
Credit History Length, of the borrower
Home Ownership, type, of the borrower
Loan Intent,
Loan Grade,
Historical Default, of the borrower

From these, we will highlight two, the Loan Grade and the Percent Income.

First, the Loan Grade is an example of a **credit rating**, an evaluation on the credit risk of the borrower. Credit Ratings are frequently used in models for their predictive power. And, estimating the probability of default and discretize it is also a technique to create them. Another example of usage of credit ratings would be leveraging the rating from a parent company for a subsidiary's loan. On the other hand, the Percent Income is an example of a **financial ratio**, which aims to quantify directly relationships in sets of financial data – in this case between the loan amount and the borrower's income –. Financial ratios are widely used in the sector and are strong predictors. As

[4]https://www.kaggle.com/datasets/laotse/credit-risk-dataset

we will see in the next section, both of them rank as the most important features for the proposed model.

To assess the observed importance obtained with the SHAP Framework, we will use another financial concept. The **default rate** measures the rate at which loans or other credit assets result in a default in a pool of borrowers. To define these pools, or bins, for continuous features we will discretize them in percentiles – considering only the 5-95 percentiles interval before segmentation – and measure the observed default rate as the ratio between the number of defaulters and the total of borrowers in each bin. For categorical features, this computing is direct.

As an example of a good sort between defaulters and good borrowers consider the percentiles distribution and default rates graph for the Percent Income feature in Figure 12.

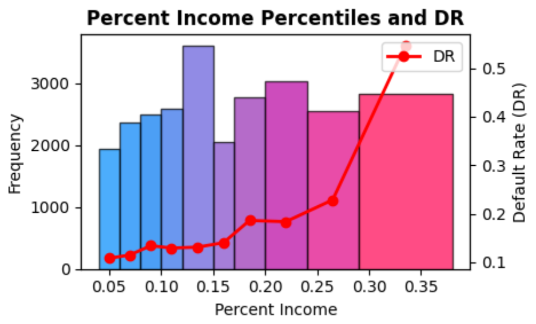


FIG. 12. Percent Income Percentiles and Default Rates.

Not only do we observe an increasing relation between the default rate and the feature – that should get reflected in the SHAP Values as seen in section 4.5 – but also a great sort between defaulters and good borrowers. Indeed, there is a .4 difference between the default rates of the first and last bins.

However, using only the observed default rate we cannot explain certain behaviors. For example, consider the distribution and default rates graph for the Home Ownership feature in Figure 13.

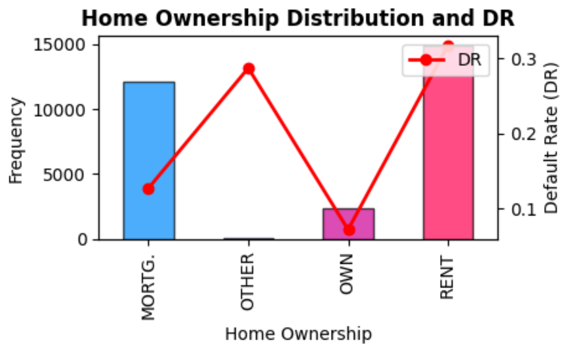


FIG. 13. Home Ownership Distribution and Default Rates.

As we will see later, the Home Ownership feature ranks as the fourth most important feature, which does not align with this poor classification of defaults – just a .2 maximum difference among observed default rates and it does not appear a monotonic relation –. To explain this case, we will consider the **Adjusted Default Rate** which is equal to the ratio between the defaulters distribution and the good borrowers distribution evaluated at the corresponding bin. This new metric is reflected in the distribution and adjusted default rates graphic for the Home Ownership feature in Figure 14.

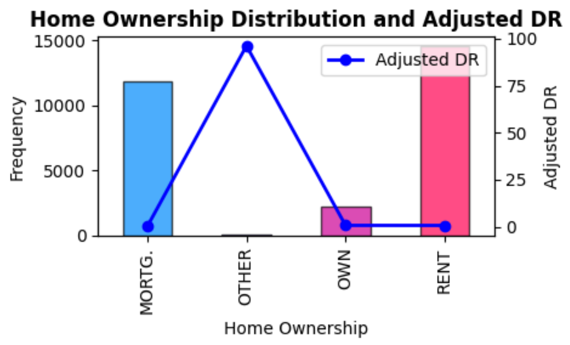


FIG. 14. Home Ownership Distribution and Adjusted Default Rates.

Now we can assign this feature's rank to the "Other" bin which serves as a financial alert for borrowers. **Financial alerts**, as one may sense, are used to anticipate great risks of default and act as early warning signals, enabling financial institutions to take proactive measures to mitigate potential losses. Other examples may be a refinanced loan, which could indicate that the borrower is struggling to meet the original terms of its loan, or a first failure to meet obligations, such as missing a payment deadline.

The annexes include both the conventional and percentiles distributions with their corresponding observed conventional and adjusted default rates for each feature. For more information about the features refer to the Kaggle Dataset repository.

In the preprocessing part, all features have been encoded – if categorical –, variance reduced via logarithms – in case of high variance – and standardized. It is not the purpose of this project to assess whether or not this is the best approach for this dataset or its explanation.

On the model part, we considered a dense neural network as the black-box model to study. Specifically, it has two hidden layers with RELU as the activation function, with 64 and 32 neurons each; and an output layer with a single neuron and a sigmoid activation function to estimate the probability of default. With this model, we obtained a precision of 87% and a recall – which quantifies the predictive power in the default label – of 61%. Again, it is not the purpose of this project to assess whether or not these particular choices are the best options to predict the probability of default. Readers interested may refer to the GitHub Repository – see Preface – for replications and more information.

5.3. Estimating SHAP Values with Deep SHAP

The selected technique to apply the SHAP Framework to this case study is Deep SHAP due to the computational cost of both SHAP and Kernel SHAP Values. Indeed, as we have $|F| = 11$ features – we used $h_x = id$ – and the cost is $\mathcal{O}(2^{|F|})$ for both of these techniques, we obtained the following times from computing the corresponding SHAP Values for a single observation in this model – refer to the GitHub Repository for replicability –.

Technique	Seconds	Minutes
SHAP	1796.9162	29.9486
Kernel SHAP	245.2316	4.0871
Deep SHAP	0.0456	0.0007

FIG. 15. Execution Times for computing the SHAP Values for each technique and a single observation.

As seen, Deep SHAP becomes the best technique to experiment with the model proposed due to its efficiency while still providing meaningful and accurate approximations of the SHAP Values.

After training the model, the observed Deep SHAP Values for a specific sample of 200 observations are shown in Figure 16 – refer to the GitHub Repository for replicability –. In this context, the Deep SHAP Values, as an approximation to SHAP Values, may be interpreted as the estimated average probability, over all possible features subsets, that each feature increases or decreases.

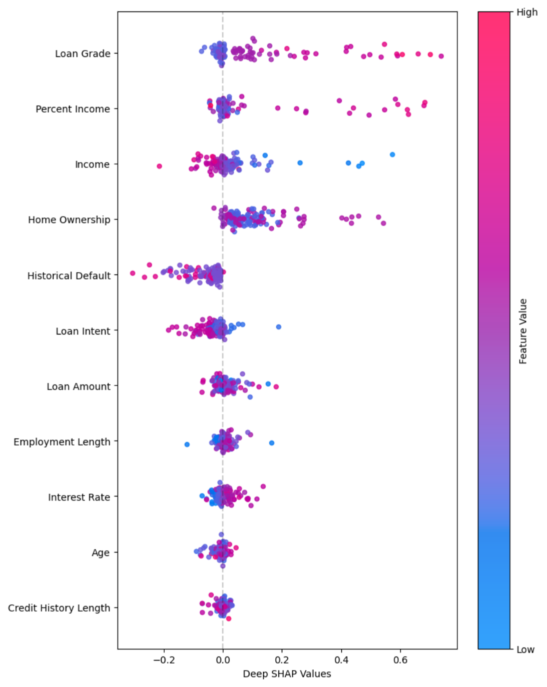


FIG. 16. Deep SHAP Values for the Credit Risk Model.

We ranked the features by the maximum observed Deep SHAP Value in absolute value. As seen, the most important features are Loan Grade and Percent Income, both with an increasing relation with the default rate; Income, with a non-increasing relation; and Home Ownership, which we clarified in the previous subsection as being used as a financial alert. The least important features are Credit History Length and Age, both of which have a poor relation with the default rate – see annexes –.

As an anomalous case, we must highlight the Interest Rate feature, which, by itself, has a great relation with the default rate. However, it ranks among the least important features even though its behavior is similar to the Percent Income feature in both the conventional and adjusted default rates – see annexes –. We could not determine the root cause of this phenomenon. Its relation with the DR is shown in Figure 17.

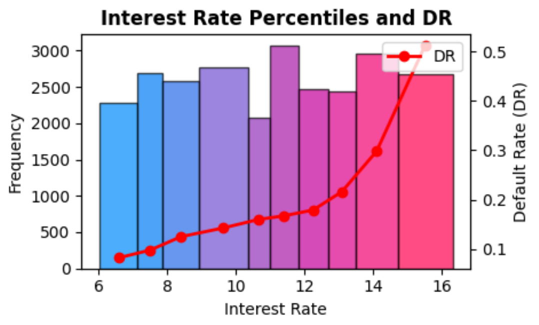


FIG. 17. Interest Rate Percentiles and Default Rates.

For an analysis for every feature we refer the reader to the annexes where we included graphs with the distributions, percentiles and its corresponding conventional and adjusted default rates.

Despite the Interest Rate feature presenting an anomaly, in general, the obtained results align well with both financial intuition and quantitative perspectives. Financial intuition refers to the understanding and expectations based on experience and established financial theories. This intuitive understanding is confirmed by the observed data and it demonstrates how effectively the model distinguishes between borrowers, reinforcing the reliability of the SHAP Values in capturing the underlying patterns.

Thus, we can conclude that the SHAP Framework is a strong approach – both from theoretical and empirical perspectives – for explainability in the credit risk field. Even though its incorporation, alongside stronger methods, across diverse sectors is already under investigation, new advancements in both explainability and traditional sectors are required. With them, which will clearly lead to more efficiency, efficacy and robustness in all these areas, Explainable Machine Learning will be able to fully exploit the latest opportunities from these black-box-models and confront the challenges that emerge along the way.

Proofs

Lemma 2.1. *Let $f : \mathbb{R}^d \to \mathbb{R}$ be the model function, $x \in \mathbb{R}^d$ the selected point from the input space and $g(z) = \beta_0 + \sum_{i=1}^d \beta_i z_i$ be the linear explainable function with intercept given by LIME when $\mathcal{L}$ is the weighted linear regression, $\pi_x = 1$ and $\Omega = 0$. Consider the sample around x, $Y = \{y_i = f(z_i) | z_i = x + \delta_i\}_{i=1}^n$, $\delta_i \sim N(0, \sigma^2 I_n)$. Then, the estimated coefficients for g verify the equation*

$$\hat{\beta} = \begin{pmatrix} f(x) - \sum_{i=1}^d \partial_i f(x) \cdot x_i \\ \nabla f(x) \end{pmatrix} + \sigma^2 C\, O(||\frac{\delta}{\sigma}||^2)\, \mathbf{1}$$

where C is a constant and $\mathbf{1}$ is the vector of ones. Thus, the function

$$g(z) \approx f(x) + \sum_{i=1}^d \partial_i f(x_i) \cdot (z_i - x_i)$$

is an approximation of the tangent plane of f in x when σ^2 is small.

Proof. Consider the linear model

$$f(z_1, ..., z_n) = \beta_0 + \beta_1(z_1 - x_1) + ... + \beta_n(z_n - x_n) + \epsilon$$

Then, with the Taylor Series expansion, we obtain that the expression for the target sample is the following.

$$Y = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} = \begin{pmatrix} f(z_{11}, ..., z_{1n}) \\ \vdots \\ f(z_{m1}, ..., z_{mn}) \end{pmatrix} = \begin{pmatrix} f(x) + \nabla f(x)\,(z_1 - x) + O(||z_1 - x||^2) \\ \vdots \\ f(x) + \nabla f(x)\,(z_m - x) + O(||z_m - x||^2) \end{pmatrix}$$

$$= X \begin{pmatrix} f(x) \\ \nabla f(x) \end{pmatrix} + O\left(||Z - X||^2\right) \mathbf{1} = X \begin{pmatrix} f(x) \\ \nabla f(x) \end{pmatrix} + O\left(||\delta||^2\right) \mathbf{1}$$

and the features sample

$$X = \begin{pmatrix} 1 & z_{11} - x_1 & \dots & z_{1n} - x_n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & z_{m1} - x_1 & \dots & z_{mn} - x_n \end{pmatrix}$$

Thus, the linear regression solution is

$$\hat{\beta} = \left(X^T X\right)^{-1} X^T Y = \left(X^T X\right)^{-1} X^T \left(X \begin{pmatrix} f(x) \\ \nabla f(x) \end{pmatrix} + O(||\delta||^2)\, \mathbf{1} \right)$$
$$= \begin{pmatrix} f(x) \\ \nabla f(x) \end{pmatrix} + \sigma^2 C\, O\left(||\frac{\delta}{\sigma}||^2 \right) \mathbf{1}$$

where C is constant. Arranging the independent terms and using that this linear solution is the one that minimizes the likelihood function we achieve the desired result.

□

This lemma and its proof are original from the author and revised by Alejandra Cabaña.

Lemma 3.1 *Consider* $\{C_{\Delta x_i \Delta y_j}\}_{i=1,j=1}^{n,m}$ *and* $\{C_{\Delta y_j \Delta z}\}_{j=1}^{m}$ *satisfying the Summation to Delta property and* $\{m_{\Delta x_i \Delta z}\}_{i=1}^{n}$ *verifying the Chain Rule for Multipliers. Then* $\{C_{\Delta x_i \Delta z}\}_{i=1}^{n}$ *follows the Summation to Delta Property.*

Proof. By a direct computation using the Std and CRfM properties along with the definition of multiplier:

$$\begin{aligned}
\sum_i C_{\Delta x_i \Delta z} &= \sum_i \Delta x_i m_{\Delta x_i \Delta z} = \sum_i \Delta x_i \sum_j m_{\Delta x_i \Delta y_j} m_{\Delta y_j \Delta z} \\
&= \sum_i \Delta x_i \sum_j \frac{C_{\Delta x_i \Delta y_j}}{\Delta x_i} m_{\Delta y_j \Delta z} = \sum_i \sum_j C_{\Delta x_i \Delta y_j} m_{\Delta y_j \Delta z} \\
&= \sum_j m_{\Delta y_j \Delta z} \sum_i C_{\Delta x_i \Delta y_j} = \sum_j m_{\Delta y_j \Delta z} \Delta y_j \\
&= \sum_j \frac{C_{\Delta y_j \Delta z}}{\Delta y_j} \Delta y_j = \sum_j C_{\Delta y_j \Delta z} = \Delta z
\end{aligned}$$

□

This proof has been extracted from the supplementary material of [4].

Theorem 4.1 (**Shapley Values**). *Let $(\mathcal{N}, \nu)$ be a cooperative game. Then there exists a unique solution $\phi : \mathbb{R}^{\mathcal{L}(\mathcal{N})} \to \mathbb{R}^n$ satisfying all Symmetry, Carrier and Linear Properties. This solution verifies the following equation*

$$\phi_i(\nu) = \sum_{S \subseteq \mathcal{N} \setminus \{i\}} \frac{|S|!\,(|\mathcal{N}| - |S| - 1)!}{|\mathcal{N}|!} \left(\nu(S \cup \{i\}) - \nu(S)\right)$$

and its values are known as Shapley Values.

Proof. We will focus on proving that there exist at most one function ϕ that satisfies the three properties as verifying that Shapley Values satisfy them is direct.

By the Linear Property, ϕ is an affine mapping from $\mathbb{R}^{\mathcal{L}}$ to $\mathbb{R}^n$. Furthermore, by the Carrier Property, if $\mathbf{0}$ is the cooperative game that assigns worth 0 to every coalition then $\phi_i(\mathbf{0}) = 0$ for every player i. Therefore, ϕ is, in fact, a linear mapping. Observe that $\mathbb{R}^{\mathcal{L}}$, the set of every cooperative game on $\mathcal{N}$, is a $(2^n - 1)$-dimensional vector space. Now, we will find a basis for it.

For any coalition R, let w_R be the cooperative game such that

$$w_R(S) = 1 \,\text{if}\; R \subseteq S, \qquad w_R(S) = 0 \text{ otherwise}$$

that is, a coalition has worth 1 in w_R if it contains all the players in R and 0 otherwise. By the Carrier Property, the payoff verifies

$$\sum_{i \in R} \phi_i(w_R) = 1, \qquad \phi_j(w_R) = 0 \;\forall j \notin R$$

as both R and $R \cup \{j\}$ are carriers of w_R. Now we will prove their linear independence. Consider the equation

$$\sum_{R \in \mathcal{L}(\mathcal{N})} \alpha_R w_R = 0$$

By reduction to the absurd, let S be a coalition of minimal size such that $\alpha_S \neq 0$. Then

$$0 = \sum_{R \in \mathcal{L}(\mathcal{N})} \alpha_R w_R(S) = \sum_{R \subseteq S, R \neq \emptyset} \alpha_R = \alpha_S$$

where the last equality is due to the minimal size of S, which is a contradiction. Thus, $\{w_R | R \in \mathcal{L}(\mathcal{N})\}$ is linearly independent in $\mathbb{R}^{\mathcal{L}(\mathcal{N})}$. In fact, as there are $2^n - 1$ such games w_R, one for each coalition R, is a basis.

Finally, by the Symmetry Property, for any w_R all players in R must get the same payoff such that

$$\phi_i(w_R) = \frac{1}{|R|},\; \forall i \in R \qquad \phi_j(w_R) = 0,\; \forall j \notin R$$

Thus, as a linear mapping is completely determined by what it does on a basis of the domain, there exists a unique linear mapping satisfying all three Symmetry, Carrier and Linear Properties. □

This proof has been extracted from [5] and it also applies to **Theorem 4.2** (**Shapley Values for AFAM**).

Theorem 4.3 (Kernel SHAP). *Consider the explainable model from Linear LIME*

$$g = \min_{h \in G} \mathcal{L}(f, h, \pi_x) + \Omega(h)$$

Then, the hyperparameters $\mathcal{L}$, π_x and Ω that retrieve the Shapley Values for AFAM are

$$\mathcal{L}(f, h, \pi_x) = \sum_{z' \in F'} \pi_{x'}(z') \, \left(f_x(z') - h(z')\right)^2$$

$$\pi_{x'}(z') = \frac{d-1}{\binom{d}{|z'|}|z'|(d-|z'|)}$$

$$\Omega(h) = 0$$

Proof. Let $X \in \mathbb{M}_{2^d \times d}(\{0,1\})$ be the matrix of all possible binary vectors of length d. Then, the coefficients of g given by the weighted linear regression are:

$$\hat{\beta} = \left(X^T W X\right)^{-1} X^T W y$$

where W is a diagonal matrix with the weights given by $\pi_{x'}$ and $y_i = f_x(S_i)$ are the output values for each row of X.

Now, to compute $X^T W X$, as $\pi_{x'}(\mathbf{0}) = \pi_{x'}(\mathbf{1}) = \infty$ consider them as a large constant a. Then $X^T W X = \frac{1}{d-1} I + cJ$ where c is a positive constant, I is the identity matrix and J is the matrix of all ones. After inverting this product and taking the limit $c \to \infty$ we obtain $(X^T W X)^{-1} = I + \frac{1}{d-1}(I - J)$. On the other hand, observe that $X^T W$ is the matrix such that all the ones in X^T have been replaced with $\pi_{x'}(z')$ and where $|z'|$ is the number of ones in that column of X^T.

Then, the coefficient $\hat{\beta}_j$ correspond to the dot product between the j'th row of $\left(X^T W X\right)^{-1} X^T W$ and y. Due to the previous paragraph, this j'th row has the following form.

$$\left(\left(X^T W X\right)^{-1} X^T W\right)_j = \left(\pi_{x'}(z'_i) \left(X_{i,j} - \frac{|z'| - X_{i,j}}{d-1}\right)\right)_{i=1}^{2^d}$$

where $X_{i,j}$ is the term (i,j) of X and z'_i is the i'th row of X. Developing this expression.

$$\begin{aligned} \pi_{x'}(z'_i) \left(X_{i,j} - \frac{|z'_i| - X_{i,j}}{d-1}\right) &= \frac{d-1}{\binom{d}{|z'|}|z'_i|(d-|z'_i|)} X_{i,j} - \frac{|z'_i| - X_{i,j}}{\binom{d}{|z'_i|}|z'_i|(d-|z'_i|)} \\ &= \frac{(d-|z'_i|-1)!(|z'_i|-1)!}{d!} \left((d-1)X_{i,j} - (|z'_i| - X_{i,j})\right) \end{aligned}$$

Therefore, when $X_{i,j} = 0$ and $X_{i,j} = 1$ we get respectively

$$-\frac{(d-|z'_i|-1)!(|z'_i|)!}{d!}, \quad \frac{(d-|z'_i|-2)!(|z'_i|-1)!}{d!}$$

Thus, taking the dot product with y we obtain

$$\hat{\beta}_j = \sum_{z' \subseteq x' \setminus \{i\}} \frac{|z'|! \, (d-|z'|-1)!}{d!} \left(f_x(z' \cup \{i\}) - f_x(z')\right) = \phi_j$$

□

This proof has been extracted from the supplementary materials of [6].

For this last proof, we must highlight the fact that we could not replicate it even though the final result is the desired. In particular, for the cases $d = 2, 3, 4$ we could not derive the equation $X^T W X = \frac{1}{d-1} I + cJ$ even though we could for $(X^T W X)^{-1} = I + \frac{1}{d-1}(I - J)$ when $c \to \infty$. Indeed, consider $d = 3$. Then, using Wolfram, we obtain

$$X = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \quad X^T W X = \begin{pmatrix} 1 + a & \frac{1}{3} + a & \frac{1}{3} + a \\ \frac{1}{3} + a & 1 + a & \frac{1}{3} + a \\ \frac{1}{3} + a & \frac{1}{3} + a & 1 + a \end{pmatrix}$$

where $W = \operatorname{diag}(a, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, a)$. And by inverting and taking the limit when $a \to \infty$

$$\lim_{a \to \infty} (X^T W X)^{-1} = \begin{pmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} & 1 \end{pmatrix}$$

Furthermore, if we invert the first equation and compute the limit when $c \to \infty$ we obtain

$$\lim_{c \to \infty} \left(\frac{1}{2} I + cJ \right)^{-1} = \begin{pmatrix} \frac{4}{3} & -\frac{2}{3} & -\frac{2}{3} \\ -\frac{2}{3} & \frac{4}{3} & -\frac{2}{3} \\ -\frac{2}{3} & -\frac{2}{3} & \frac{4}{3} \end{pmatrix}$$

As seen, the equation for $X^T W X$ does not hold while the one for $(X^T W X)^{-1}$ does. We could not derive the right formula. As mentioned, this is a minor error as the theorem is true due to the veracity of the last equation and it can be seen in the comparative of techniques from subsection 4.5.

Corollary 4.1 (**Linear SHAP**). *Let the model function be linear $f(z) = \beta_0 + \sum_{i=1}^{d} \beta_i z_i$. Then, the SHAP Values are*

$$\phi_0 = \beta_0 + \sum_{i=1}^{d} \beta_i \mathbb{E}(z_i)$$

$$\phi_i = \beta_i \left(x_i - \mathbb{E}(z_i)\right)$$

Proof. As SHAP Values use as characteristic function $f_x = E\left(f \mid x_S\right)$, then

$$\phi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!\,(|F| - |S| - 1)!}{|F|!} \left(\mathbb{E}\left(f \mid x_{S \cup \{i\}}\right) - \mathbb{E}\left(f \mid x_S\right)\right)$$

We already demonstrated in subsection 4.4 that if f is linear then $E\left(f \mid z_S\right) = f\left([z_S, \mathbb{E}\left(z_{\overline{S}}\right)]\right)$. Therefore

$$= \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!\,(|F| - |S| - 1)!}{|F|!} \left(\beta_i x_i - \beta_i \mathbb{E}\left(z_i\right)\right) = \beta_i \left(x_i - \mathbb{E}\left(z_i\right)\right)$$

where we used

$$\sum_{S \subseteq F \setminus \{i\}} \frac{|S|!\,(|F| - |S| - 1)!}{|F|!} = 1$$

which may be deduced from the interpretation for Shapley Values from subsection 4.2.

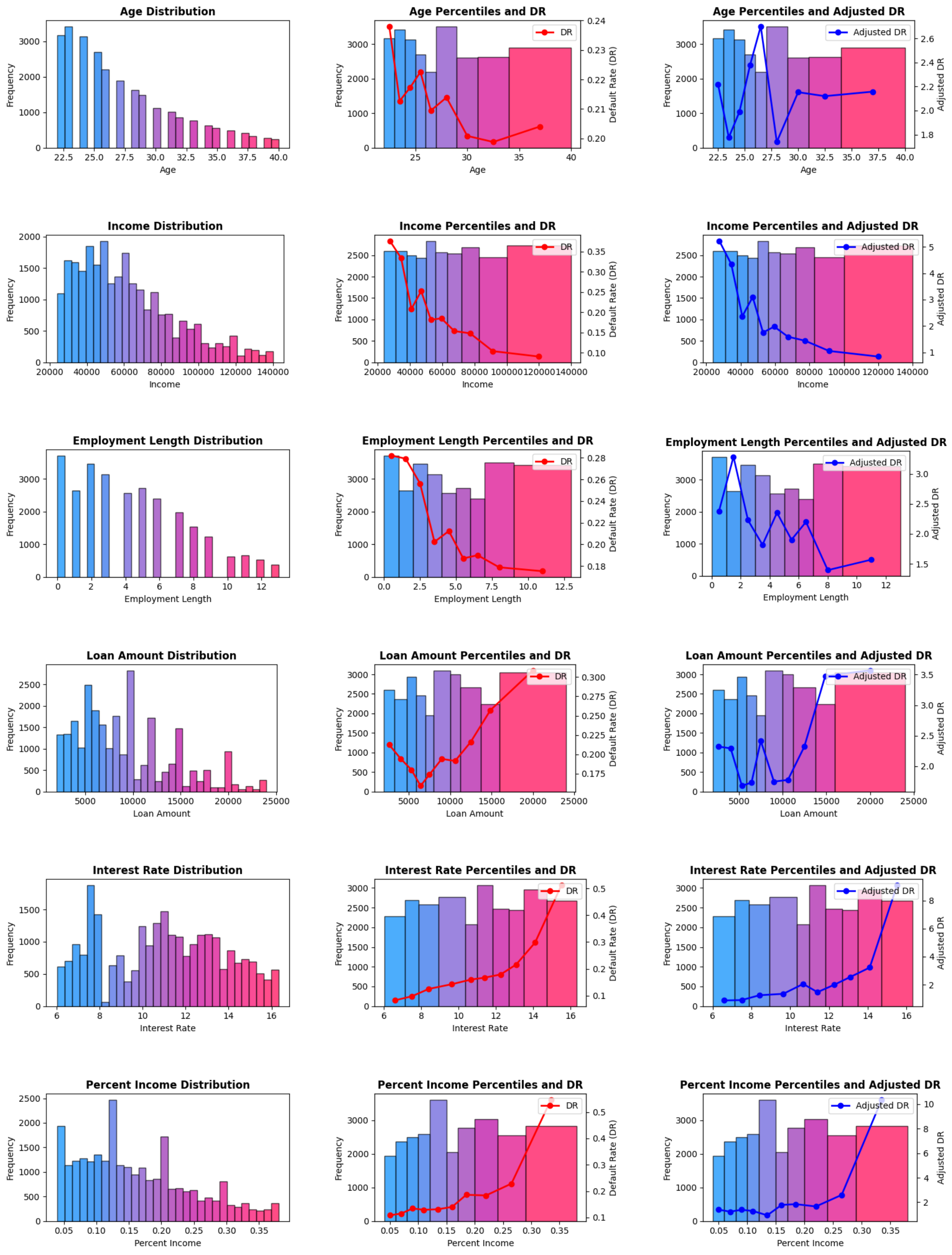
For ϕ_0 the computation is also direct as

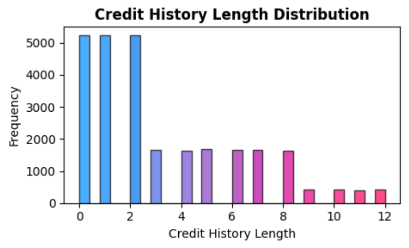
$$\phi_0 = \mathbb{E}(f | x_{\emptyset}) = \beta_0 + \sum_{i=1}^{d} \beta_i \mathbb{E}(z_i)$$

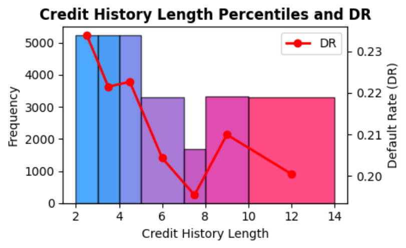
$\square$

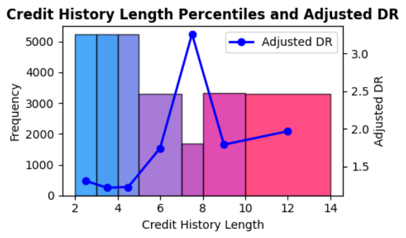
This corollary is stated in [6] but without further details. Here we present a resolution. Also, we corrected the original presentation which enunciated $\phi_0 = \beta_0$. This statement assumes all features are centered in 0.

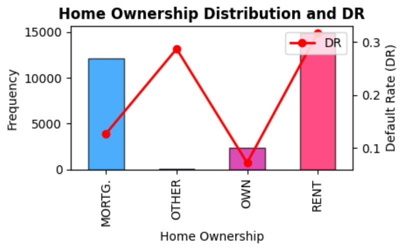
Credit Risk Model Features Analysis

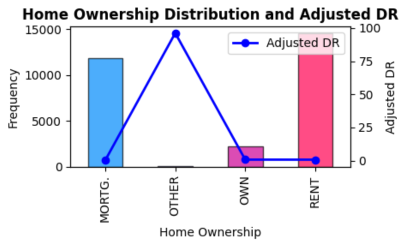


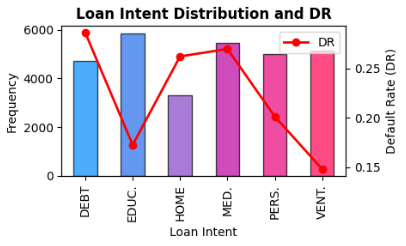

Credit History Length Distribution
Frequency
Credit History Length
0
1000
2000
3000
4000
5000
0
2
4
6
8
10
12

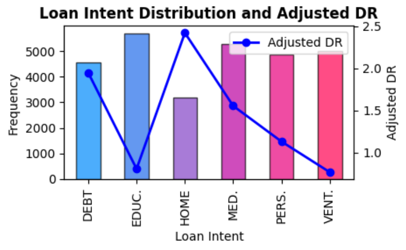

Credit History Length Percentiles and DR
DR
Frequency
Default Rate (DR)
Credit History Length
0
1000
2000
3000
4000
5000
0.20
0.21
0.22
0.23
2
4
6
8
10
12
14

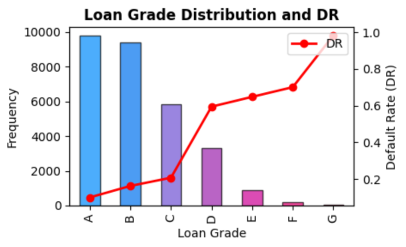

Credit History Length Percentiles and Adjusted DR
Adjusted DR
Frequency
Adjusted DR
Credit History Length
0
1000
2000
3000
4000
5000
1.5
2.0
2.5
3.0
2
4
6
8
10
12
14

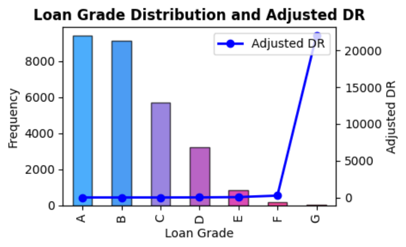

Home Ownership Distribution and DR
DR
Frequency
Default Rate (DR)
Home Ownership
0
5000
10000
15000
0.1
0.2
0.3
MORTG.
OTHER
OWN
RENT

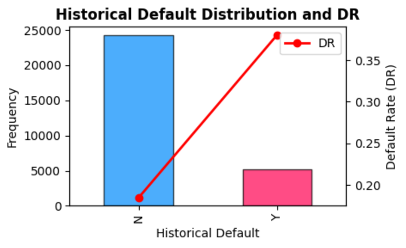

Home Ownership Distribution and Adjusted DR
Adjusted DR
Frequency
Adjusted DR
Home Ownership
0
5000
10000
15000
0
25
50
75
100
MORTG.
OTHER
OWN
RENT

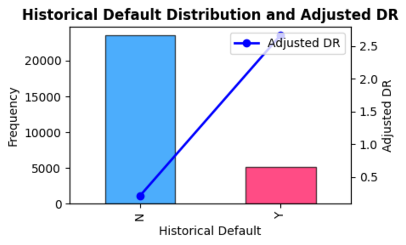

Loan Intent Distribution and DR
DR
Frequency
Default Rate (DR)
Loan Intent
0
2000
4000
6000
0.15
0.20
0.25
DEBT
EDUC.
HOME
MED.
PERS.
VENT.


Loan Intent Distribution and Adjusted DR
Adjusted DR
Frequency
Adjusted DR
Loan Intent
0
1000
2000
3000
4000
5000
1.0
1.5
2.0
2.5
DEBT
EDUC.
HOME
MED.
PERS.
VENT.


Loan Grade Distribution and DR
DR
Frequency
Default Rate (DR)
Loan Grade
0
2000
4000
6000
8000
10000
0.2
0.4
0.6
0.8
1.0
A
B
C
D
E
F
G


Loan Grade Distribution and Adjusted DR
Adjusted DR
Frequency
Adjusted DR
Loan Grade
0
2000
4000
6000
8000
0
5000
10000
15000
20000
A
B
C
D
E
F
G


Historical Default Distribution and DR
DR
Frequency
Default Rate (DR)
Historical Default
0
5000
10000
15000
20000
25000
0.20
0.25
0.30
0.35
N
Y


Historical Default Distribution and Adjusted DR
Adjusted DR
Frequency
Adjusted DR
Historical Default
0
5000
10000
15000
20000
0.5
1.0
1.5
2.0
2.5
N
Y

```
def get_SHAP_values(model, data, data_x, x0):
  #@model: A keras model
  #@data: An array of observations where \phi_0 is estimated
  #@data_x: An array of observations whose SHAP Values will be computed
  #@x_0: An observation with the expected value for each feature.

  from math import factorial as fact

  def get_binary_subsets(n):#Gets every binary array of length n
    secuencias = [format(i, f'0{n}b') for i in range(2 ** n)]
    secuencias_np = [np.array(list(map(int, list(seq)))) for seq in secuencias]
    return secuencias_np

  def weight(S):#Computes the weight from the Shapley Values Formula given the
   coallition S
    return fact(np.sum(S)) * fact(n - np.sum(S) - 1) / fact(n)

  def get_shap(i):#Computes the SHAP Value for the player i
    one = np.ones(n)

    subsetsF_diff_i = [cadena for cadena in subsetsF if cadena[i] == 0]#Every
   subset of F that does not contain i

    #SHAP Values Formula
    shap_i = 0
    for S in subsetsF_diff_i:
      S_plus_i = np.copy(S)
      S_plus_i[i] = 1

      if(np.all(S == 0)):
        f_S = expected_value
      else:
        f_S = model.predict([np.multiply(S, x) + np.multiply(one - S, [x0])],
   verbose = 0)[0][0]
        #np.multiply(S, x) + np.multiply(one - S, x0) is equivalent to E(f|x_S)
    where f is linearly aproximated

      f_S_plus_i = model.predict([np.multiply(S_plus_i, x) + np.multiply(one -
   S_plus_i, [x0])], verbose = 0)

      shap_i = shap_i + weight(S) * (f_S_plus_i - f_S)
    return(shap_i[0][0])

  expected_value = np.mean(model.predict(data, verbose = 0))
  n = model.input_shape[1] #number of features

  subsetsF = get_binary_subsets(n)#Gets every subset of F

  shap_values = []
  for x in data_x:
    shap_values_x = list(map(get_shap, list(range(n))))
    shap_values.append(shap_values_x)
  return(shap_values)
```

Listing 1. SHAP Values Algorithm Implementation.

```python
def get_Kernel_SHAP_values(model, data, data_x, x0):
  #@model: A keras model
  #@data: An array of observations where \phi_0 is estimated
  #@data_x: An array of observations whose Kernel SHAP Values will be computed
  #@x_0: An observation with the expected value for each feature.

  from scipy.special import comb
  from sklearn.linear_model import LinearRegression

  def model_expected_function(model, x, x0, S):
    one = np.ones(len(S))
    x_S = list(np.multiply(S, x) + np.multiply(one - S, x0))
    return model.predict([x_S], verbose=0)[0][0]

  def get_binary_subsets(n):#Gets every binary array of length n
    secuencias = [format(i, f'0{n}b') for i in range(2 ** n)]
    secuencias_np = [np.array(list(map(int, list(seq)))) for seq in secuencias]
    return secuencias_np

  n = model.input_shape[1] #number of features
  expected_value = np.mean(model.predict(data, verbose = 0))
  subsetsF = get_binary_subsets(n)[1:-1] #every subset excep np.zero and np.one
    as they get an infinite weight

  kernel_shap_values = []
  for x in data_x:
    x = list(x)

    X_kernel = []
    Y_kernel = []
    W = []
    f_x = model.predict([x], verbose = 0)[0][0]

    for S in subsetsF:#Kernel SHAP Sample
      Y_kernel.append(model_expected_function(model, x, x0, S) - f_x * S[0] +
   expected_value * (S[0] - 1))

      X_obs = []
      for i in list(range(n))[1:]:
        X_obs.append(S[i] - S[0])
      X_kernel.append(X_obs)

      n_z = sum(S)
      W.append((n - 1 ) / (comb(n, n_z) * n_z * (n - n_z)))

    modelo = LinearRegression(fit_intercept=False)
    modelo.fit(X_kernel, Y_kernel, sample_weight = W)

    kernel_shap_values_x = np.insert(np.copy(modelo.coef_), 0, f_x -
   expected_value - sum(modelo.coef_))
    kernel_shap_values.append(list(kernel_shap_values_x))

  return kernel_shap_values
```

Listing 2. Kernel SHAP Values Algorithm Implementation

```python
def get_Deep_SHAP_Values(model, data_x, x0):
  #@model: A keras model
  #@data_x: An array of observations whose Deep SHAP Values will be computed
  #@x_0: An observation with the reference value

  def get_neurons_output(x):
    i_x = x
    neurons_output = [np.array(i_x)]
    for i in list(range(n_layers)):
      i_layer = model.layers[i] #current layer
      i_bias = i_layer.weights[1] #shape = n_inputs x 1
      i_weights = i_layer.weights[0] #shape = n_neurons x n_inputs

      #computes the neurons output
      i_linear_output = i_bias + np.dot(np.transpose(i_weights),i_x)
      neurons_output.append(np.array(i_linear_output))
      i_output = i_layer.activation(i_linear_output)
      i_x = np.copy(i_output)

      neurons_output.append(np.array(i_x))
    return neurons_output #neurons_output.length == 2*n_layers, as for every layer it saves the linear and non linear outputs

  n_layers = len(model.layers) #does not count the input layer
  contribution_scores = []

  #FORWARD PASS
  neurons_output_x0 = get_neurons_output(x0)
  for x in data_x:
    neurons_output_x = get_neurons_output(x)
    diff_from_ref = []
    for array1, array2 in zip(neurons_output_x, neurons_output_x0):
      resta = array1 - array2
      diff_from_ref.append(resta)

    #BACKPROPAGATION PASS
    i = len(diff_from_ref) - 1 #last layer after activation function
    C_in_out = diff_from_ref[i]
    m_in_out = [1.]

    while(i >= 1):
      if(i % 2 == 0):#activation iteration
          m_in_out = (diff_from_ref[i] / diff_from_ref[i - 1]) * np.transpose(m_in_out)#Chain Rule for Multipliers, one input
          #C_in_out does not change in this step
      else:#linear iteration
          i_layer_weigths = model.layers[int(i/2)].weights[0]#weights of the current layer

          m_in_out = np.dot(i_layer_weigths, np.transpose(m_in_out))#CRfM
          C_in_out = diff_from_ref[i-1] * np.transpose(m_in_out)#Multiplier def

     i = i - 1
    contribution_scores.append(list(C_in_out))
  return contribution_scores
```

Listing 3. Deep SHAP Values Algorithm Implementation

References

[1] A. Barredo, N. Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil, D. Molina, R. Benjamins, R. Chatila and F. Herrera. Explainable Artifical Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI. In *Information Fusion*, Volume 58, Pages 82-115, 2020.

A great paper for introducing readers to explainability and its main protagonists.

[2] M. Ribeiro, S. Singh, and C. Guestrin. Why should I trust you? Explaining the predictions of any classifier. In *arXiv*, 1602.04938v3, 2016.

The paper introducing LIME.

[3] G. Visani. LIME: explain Machine Learning predictions. Intuition and Geometrical Interpretation of LIME. In *TowardsDataScience.com*, https://towardsdatascience.com/lime-explain-machine-learning-predictions-af8f18189bfe, 2020.

While exploring the geometrical interpretation of LIME we found out this article on the subject.

[4] A. Shrikumar, P. Greenside, and A. Kundaje. Learning Important Features Through Propagating Activation Differences. In *arXiv*, 1704.02685v2, 2019.

The paper introducing DeepLIFT.

[5] R. Myerson. Game Theory: Analysis of Conflict. In *Harvard University Press*, 1991.

An essential introduction to Game Theory. The fundamentals here presented correspond to Chapter 9, Coalitions in Cooperative Games.

[6] S. Lundberg, and S. Lee. A Unified Approach to Interpreting Model Predictions. In *31st Conference on Neural Information Processing Systems (NIPS)*, 2017.

The paper introducing the SHAP Framework.

[7] J. Hull. FRM Exam Part I, Financial Markets and Products. In *Global Association of Risk Professionals (GARP)*, 2023.

Preparatory material for the Financial Risk Manager Certification from GARP.

[8] D. Esposito, M. Carminati, M. Musto, M. Amorese, and M. Cecchin. Credit Risk: Basel IV Regulatory Framework and New Frontiers. In *Iason Ltd Research Paper Series*, 2022.

A research paper addressing the latest Basel IV regulation from the Basel Committee and its repercussions on applied Machine Learning. Thanks to Lluis Vidal from Banco Sabadell for referring me to it.

[9] European Parliament. On Prudential Requirements for Credit Institutions and Investment Firms and Amending Regulation. In *Official Journal of the European Union*, Regulation (EU) No 575 / 2013, 2013.

The official legislation from the European Parliament containing the current regulation for credit institutions.