XAI

Emanuele Ciccarelli, Michele Viscione

1 Introduction

Artificial Intelligence (AI) based algorithms, especially Deep Neural Networks' ones, are transforming the way we approach real-world tasks usually done by humans. Recent years have seen a surge in the use of Machine Learning (ML) algorithms in automating various facets of the science, business, and social workflow [13].
Despite their widespread adoption, machine learning models remain mostly black boxes. Understanding the reasons behind predictions is, however, quite important in assessing trust, which is fundamental if one plans to take action based on a prediction, or when choosing whether to deploy a new model with respect to old one. Such understanding also provides insights into the model, which can be used to transform an untrustworthy model or prediction into a trustworthy one [1].
In some applications, simple models (e.g., linear models) are often preferred for their ease of interpretation, even if they may be less accurate than complex ones. However, the growing availability of big data has increased the benefits of using complex models, so bringing to the forefront the trade-off between accuracy and interpretability of a model's output [3].
It is here where Explainable Artificial Intelligence (XAI) becomes valuable. XAI is a field of Artificial Intelligence (AI) that promotes a set of tools, techniques, and algorithms that can generate high-quality interpretable, intuitive, human-understandable explanations of AI decisions [13], while maintaining high performance levels.
In section 2, we present a general taxonomy for XAI while in section 3, we explore some XAI algorithms. Finally, in section 4 we discuss, only from an high level point of view, the field concerning the evaluation of XAI methods.

2 XAI Taxonomy

In this section we will present a widely cited taxonomy in the XAI field, that is the one illustrated in [13]. This, of course, is not the only taxonomy that was proposed in the XAI context, for example in [15] the authors presented another taxonomy that might be helpful in the near future. However the taxonomy presented in [13] is perhaps the most popular in the community of XAI as well as the most accurate and useful for what we have analyzed so far. With this approach we can classify the XAI methods according to the following three criteria:

2.1 Scope

The Scope of explanations can be either local or global but some methods can be extended to both.

- Locally explainable methods are designed to express the reasons for a specific decision or single prediction. Therefore, this kind of explainability is often used to generate an individual explanation, generally to justify why the model made a specific decision for an instance.
- Globally explainable models provide insight into the decision of the model as a whole - leading to an understanding of all the different possible outcomes.

Arguably, global model interpretability is hard to achieve, especially for models that exceed a handful of parameters. Analogously to human, who focus their effort on only part of the model in order to comprehend the whole of it, local interpretability can be more readily applicable [19]. Depending on the type of scope of our explainability method we have the following possibilities:

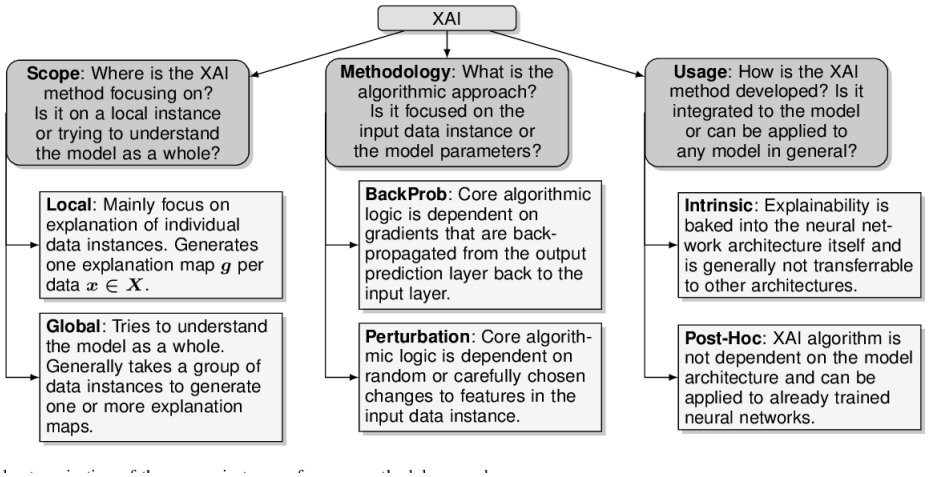

Figure 1: A scheme summary of the illustrated taxonomy on XAI

1. We can use the standard global model interpretability in order to answer how the model makes predictions.
2. Global model interpretability can also be used in a modular way in order to identify how do parts of the model influence predictions.
3. We can use local interpretability for a group of predictions to understand why did the model make specific decisions for a group of instances.
4. Finally, the standard local interpretability is used on a single prediction to justify why the model makes a specific decision for an instance.

An interesting observation pointed out in [19] is that in the current literature, local explanations are the most used methods to generate explanations in DNNs.

2.2 Methodology

With methodology we intend the categorization of a XAI method based on the methodology of implementation. In general, explainable algorithms can be classified into these two class:

- Backpropagation-based methods. in which the explainable algorithm does one or more forward pass through the neural network and generates attributions during the backpropagation stage utilizing partial derivatives of the activations.
- Perturbation-based methods, they are algorithms inn which the main focus is the perturbation of the feature set of a given input instance by either using occlusion, partially substituting features with filling operations or generative algorithms, masking, conditional sampling, etc. Here, generally, only forward pass is enough to generate the attribution representations without the need for backpropagating gradients.

2.3 Usage

Another important way to classify model interpretability techniques is whether they are model agnostic, meaning that they can be applied to any types of ML algorithms, or model intrinsic, meaning that they are dependent on the model architecture.

- Model-intrinsic category is constituted by all explainable algorithms which are dependent on the model architecture. Most model-intrinsic algorithms are model-specific such that any change in the architecture will need significant changes in the method itself.

- Model-agnostic methods instead are not tied to a particular type of ML model. In other words, this class of methods separates prediction from explanation. Model-agnostic interpretations are usually post-hoc. Significant research interest is seen in developing model-agnostic post-hoc explanations.

3 XAI Algorithms

3.1 LIME

What is LIME?

LIME stands for Local Interpretable Model-agnostic Explanations and it is an implementation of a local surrogate model. The latter is an interpretable model that is used to explain individual predictions of black box machine learning models. The goal of LIME is to explain the predictions of any classifier in an interpretable and faithful manner, by learning an interpretable model locally around the prediction.

Before moving on, we present some important concepts that we will use in the construction of the *explanations*. In particular, we want to clarify the fundamental distinction between features and interpretable data representations. The features are the input of the original model, while the interpretable data representations are the input used by the explanation and must be understandable to humans. The few following example can further explain this concept:

- A possible interpretable representation for text classification is a binary vector indicating the presence or absence of a word, even though the classifier may use more complex (and incomprehensible) features such as word embeddings.

- For image classification, an interpretable representation may be a binary vector indicating the "presence" or "absence" of a contiguous patch of similar pixels (a super-pixel), while the model may be classifying on the image as a tensor with three color channels per pixel.

In our discussion we denote with $x \in \mathbf{R}^d$ the original representation of an instance being explained, and we use $x' \in \{0,1\}^{d'}$ to denote a binary vector for its interpretable representation.

How do we obtain *explanations*?

Before presenting the formal procedure on how LIME produces explanation we want first to give the general idea behind the algorithm. For now, forget about the training data and imagine that you only have a black box (the machine learning algorithm) to which you can give the input data points and obtain the predictions of the model as the output. The black box approximation is necessary since we want to build an explainer that is model-agnostic. You can probe the box as often as you want by checking the input-output pairs. In fact, this is the reason why we said to temporarily forget about training data. Our goal is to understand why the machine learning model made a certain prediction. Now, to solve this problem, LIME tests what happens to the predictions when you give variations of your data into the machine learning model. In order to do that LIME generates a new data set by means of perturbing the original data points. The machine learning model that we want to explain is then fed with this new training set. This new dataset elements are weighted by their proximity to the instance of interest and will LIME trains an interpretable model over it to achieve its goal.

Now, we will formalize this idea, Formally, we define an explanation as a model $g \in G$ where G is a class of potentially interpretable models, such as linear models, decision trees, or falling rule lists. The domain of g is $\{0,1\}^{d'}$, i.e. g acts over absence/presence of the interpretable components. Even though g is an interpretable model this is not sufficient alone since it could still be too difficult to be humanly understandable. For instance, think about a decision tree. In general this model is prone to be fully interpretable, but what happens when the tree is too deep? At a certain point we would lose

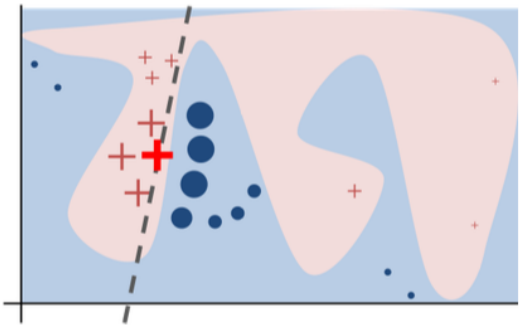

Figure 2: Toy example to present intuition for LIME. The black-box model's complex decision function f (unknown to LIME) is represented by the blue/pink background, which cannot be approximated well by a linear model. The bold red cross is the instance being explained. LIME samples instances, gets predictions using f, and weighs them by the proximity to the instance being explained (represented here by size). The dashed line is the learned explanation that is locally (but not globally) faithful.

track of why we are making those specific choices along the tree path, i.e. a loss of interpretability.
In order to keep track of this model "complexity" and be able to limit it as much as possible we introduce a measure of the complexity $\Omega(g)$ for a given explanation $g \in G$ (as opposed to interpretability). As an example, for decision trees $\Omega(g)$ may be the depth of the tree, as mentioned before, while for linear models $\Omega(g)$ may be the number of non-zero weights.
Let the model being explained be denoted as $f : \mathbf{R}^d \rightarrow \mathbf{R}$. We further use $\pi_x(z)$ as a proximity measure of an instance z with respect to x, so as to define locality around x. Finally, let $\mathcal{L}(f, g, \pi_x)$ be a measure of how unfaithful g is in approximating f within the locality defined by π_x.
We would like to ensure both interpretability and local fidelity, hence what we want is to minimize $\mathcal{L}(f, g, \pi_x)$ while having $\Omega(g)$ be low enough to be interpretable by humans. So, the explanation given by LIME is obtain by:

$$\xi(x) = \arg \min_{g \in G} \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{1}$$

Respecting our premises on the generality of this approach, this formulation can be used with different explanation families G, fidelity functions $\mathcal{L}$, and complexity measures $\Omega(g)$.

How do we implement it?

In the above formulation of LIME, we avoid to mention one thing that here is clarified, namely as the definition of LIME itself says, the algorithm must produce an explanation model that is model-agnostic, i.e. we cannot make any assumptions about f or, in other words, we have to treat f as a black box. So a natural question arises, how do we minimize $\mathcal{L}(f, g, \pi_x)$?
The solution implemented by LIME is to approximate $\mathcal{L}(f, g, \pi_x)$ by drawing samples, weighted by π_x. In particular, to train our explanation g we will create a training set by means of random perturbations of x'. In particular, we sample instances around x' by drawing nonzero elements of x' uniformly at random. We label this perturbed sample as $z' \in \{0, 1\}^{d'}$ (which contains a fraction of the nonzero elements of x'), then we recover the sample in the original representation $z \in \mathbf{R}^d$ and obtain $f(z)$ by giving these values to the model f. These samples are then weighted by π_x in order to ensure locally faithful. We denote this training set, obtained by perturbing x', as $\mathcal{Z}$. Given this dataset and labels, equation 1 is optimized to learn the explanation model.
It is worth to highlight that weighting the samples by π_x not only guarantees local faithfulness but also adds robustness with respect to the sampling noise and outliers.

Example

We now show an high level example, in order to better visualize the concept before discussed. The example is a decision boundary illustrated in Figure 2. The black-box model's decision function f

(unknown to LIME) is represented by the blue/pink background, which cannot be approximated well by a linear model. The bold red cross is the instance being explained. LIME samples instances and get their predictions using f, then it weights them by the proximity to the instance being explained. The dashed line is the learned explanation that is locally (but not globally) faithful. In the figure, the red crosses, with the exception of the bold cross, and the blue points are the points obtained by the sampling process of perturbing the point being explained. The size of this elements represents their relative weights (i.e. proximity). Notice that in the samples we get points that cannot be explained by the linear model as the blue points to the left corner of the image and the red cross on the right. However here we can clearly see the fact that weighting the samples by π_x adds robustness to sampling noise. In fact, this points, have a relative small importance in the training of the explanation g with respect to the points closer to the instance we want to explain.

Further examples in real contexts are shown in 1 along with a global extension of LIME (SP-LIME).

3.2 SHAP

What are Shapley values?

The Shapley value is a solution concept from cooperative game theory proposed by Lloyd Shapley in 1953. The setup is as follows: a coalition of players cooperates, and obtains a certain overall gain (payout) from that cooperation. In this contest Shapley values tells us how to fairly distribute the payout. We believe that the most effective way to explain shapley values is to first give an introductory example and then formalize it. For this purpose we will report the example given in [2]:

Example (from google whitepaper)

In this setup we have a company with a set N made of three employees $\{A, B, C\}$ (the "participants"), and a profit outcome $v(N)$. Let's say we want to distribute profit among the 3 employees based on how much they contributed to achieving that outcome. The Shapley value gives us a framework for doing so. It considers all the ways that adding each employee improved the profit compared to not having those employees. It then assigns value correspondingly.

Now let's say we have the following profit outcomes for different combinations of employees N:

- $v(\{\}) = 0$
- $v(\{A\}) = 10$
- $v(\{B\}) = 20$
- $v(\{C\}) = 30$
- $v(\{A, B\}) = 60$
- $v(\{B, C\}) = 70$
- $v(\{A, C\}) = 90$
- $v(\{A, B, C\}) = 100$

So, when the company had no employees, its profit was 0. With just A or B as employees, its profit was 10 and 20, respectively. In this example, It can be seen that the profit generally grows as we add more employees. We notice that with just employee A, the company had a profit of 10 and adding B raised the profit to 60. Does that mean B alone contributed 50 to the profit? The answer is clearly not, indeed adding B when C was the only employee only increased the profit by 40 (70 - 30).
To account for this, the Shapley value considers all the permutations of employees joining the company. We first compute all unique ways of growing the company from 0 to 3 employees:

1. $\{\} \rightarrow \{A\} \rightarrow \{A, B\} \rightarrow \{A, B, C\}$

2. $\{\} \to \{A\} \to \{A, C\} \to \{A, B, C\}$
3. $\{\} \to \{B\} \to \{A, B\} \to \{A, B.C\}$
4. $\{\} \to \{B\} \to \{B, C\} \to \{A, B.C\}$
5. $\{\} \to \{C\} \to \{B, C\} \to \{A, B.C\}$
6. $\{\} \to \{C\} \to \{A, C\} \to \{A, B.C\}$

Each time we add an employee, we compute a credit for that employee based on how much the profit grows after they were added. The path (1) gives us these credits to each employee:

- A: $v(\{A\}) - v(\{\}) = 10 - 0 = 10$
- B: $v(\{A, B\}) - v(\{A\}) = 60 - 10 = 50$
- C: $v(\{A, B, C\}) - v(\{A, B\}) = 100 - 60 = 40$

Finally, we need to compute the credits for each possible path, then average the results together.

1. $\{\} \to \{A\} \to \{A, B\} \to \{A, B, C\} \quad || \quad A = 10,\ B = 50;\ C = 40$
2. $\{\} \to \{A\} \to \{A, C\} \to \{A, B, C\} \quad || \quad A = 10,\ B = 10;\ C = 80$
3. $\{\} \to \{B\} \to \{A, B\} \to \{A, B.C\} \quad || \quad A = 40,\ B = 20;\ C = 40$
4. $\{\} \to \{B\} \to \{B, C\} \to \{A, B.C\} \quad || \quad A = 30,\ B = 20;\ C = 50$
5. $\{\} \to \{C\} \to \{B, C\} \to \{A, B.C\} \quad || \quad A = 30,\ B = 40;\ C = 30$
6. $\{\} \to \{C\} \to \{A, C\} \to \{A, B.C\} \quad || \quad A = 60,\ B = 10;\ C = 30$

By Averaging these results, we obtain the Shapley values: $A_{avg} = 30, B_{avg} = 25, C_{avg} = 45$

Formal definition of Shapley values

Before we properly formalize the Shapley values we have to move this method within the framework of machine learning. In our contest, the "game" is the prediction task for a single instance of the dataset. The "gain" is the actual prediction for this instance minus the average prediction over all instances. The "players" (or "employees") are the feature values of the instance that collaborate to receive the gain (that is to predict a certain value).
The Shapley value are defined via a value function *val* of the players in a set S, that maps subsets of players to the real numbers: $val : 2^N \to \mathbf{R}$, with $v(\emptyset) = 0$. If we denote with F the set of all possible players (features) and with S a coalition of players (set of features), then $v(s)$ describes the total expected sum of payoffs that the members of S can obtain by cooperation.
Now, with the above setting and remember that the Shapley value of a feature value is its contribution to the payout, weighted and summed over all possible feature value combinations we obtain:

$$\phi_i = \sum_{S \subseteq F \setminus \{i\}} = \frac{|S|!(|F| - |S| - 1)!}{|F|!}(val(S \cup \{i\}) - val(S)) \tag{2}$$

Properties

The Shapley value is the only attribution method that satisfies the properties of Efficiency, Symmetry, Dummy and Additivity, which together can be considered a definition of a fair payout.

Efficiency (also known as completeness): The feature contributions must add up to the difference of the prediction for an instance x and the average (i.e. in our context the overall gain).

$$\sum_{i=1}^{p} \phi_i = \hat{f}(x) - E_X(\hat{f}(X))$$

Where $\hat{f}(x)$ is the prediction of the machine learning model and $E_X(\hat{f}(X))$ is the expectation of $\hat{f}$ for the given dataset X.

Symmetry: The contributions of two feature values j and k should be the same if they contribute equally to all possible coalitions.

$$val(S \cup \{i\}) = val(S \cup \{j\}), \forall S \subseteq F \setminus \{i, j\} \Rightarrow \phi_i = \phi_j$$

Dummy: A feature j that does not change the predicted value – regardless of which coalition of feature values it is added to – should have a Shapley value of 0.

$$val(S \cup \{j\}) = val(S), \forall S \subseteq F \Rightarrow \phi_j = 0$$

Additivity: For two outcome functions, the Shapley values of their sum should be equal to the sum of the Shapley values of the two outcome functions.

These are all desirable properties for an attribution method. Notice that the property of Additivity allows us to extend the concept of Shapley value even to the ensemble models, for example by training a random forest, where the prediction is an average of many decision trees. The Additivity property guarantees that for a feature value, you can calculate the Shapley value for each tree individually, average them, and get the Shapley value for the feature value for the random forest.

Pros and cons

One of the main advantages of Shapley values is that thanks to the efficiency property, it guarantees that the difference between the prediction and the average prediction is fairly distributed among the feature values of the instance. Moreover, Shapley value is the only explanation method with a solid theory. The axioms – efficiency, symmetry, dummy, additivity – give the explanation a reasonable foundation.
The main disadvantage is that with the above definition of Shapley value, it is an unfeasible method due to its computational cost. Indeed, An exact computation of the Shapley value requires to consider 2^k coalitions, where k is the number of features. So the "absence" of a feature (like in 2), in the machine learning context, has to be simulated by drawing random instances, which increases the variance for the estimate of the Shapley values estimation. This leads, in most cases, only to an approximated but feasible solution.

From shapley values to SHAP

SHAP (SHapley Additive exPlanations) by Lundberg and Lee (2017) [3] is an explanation method based on the game theoretically optimal Shapley values. In SHAP authors proposed KernelSHAP, an alternative, kernel-based estimation approach for Shapley values inspired by local surrogate models.
The goal of SHAP is to explain the prediction of an instance x by computing the contribution of each feature to the prediction. In order to do this, SHAP explanation method computes Shapley values from coalitional game theory (previously explained). The feature values of a data instance act as players in a coalition. A player can also be a group of feature values. For instance, to explain an image pixels can be grouped to superpixels.

A novel aspect of SHAP is to see Shapley value explanation as an additive feature attribution linear method. This highlights the connection between LIME and Shapley values.
We report here some terminology that we will use in our examination of SHAP. We denote with f the original prediction model to be explained and with g a proposed explanation model. Explanation models often use simplified inputs that we will label as x' that map to the original inputs through a mapping function $x = h_x(x')$. In our framework SHAP will focus on a local method, that means, that it will try to ensure $g(z') \approx f(h_x(z'))$ whenever $z' \approx x'$ (note that $h(x') = x$ even though x' may

contain less information than x because h_x is specific to the current input x and hence contains some information itself).

SHAP define an additive feature attribution method as explanation model that is a linear function of binary variables:

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i \tag{3}$$

Where $z' \in \{0,1\}^M$, M is the number of simplified input features, and $\phi_i \in \mathbf{R}$. It may be helpful to think of the "simplified input" as a "coalition vector" where the z's describe coalitions: In a coalition vector, an entry of 1 means that the corresponding feature value is "present" and 0 that it is "absent".

Methods with explanation models matching this definition, attribute an effect ϕ_i to each feature, and summing the effects of all feature attributions approximates the output $f(x)$ of the original model. Many current methods match this definition like the precedent defined LIME and Shapley value.

Properties

Shapley values are the only solution that satisfies properties of Efficiency, Symmetry, Dummy and Additivity. since SHAP computes Shapley values, it also satisfies these, but in SHAP these properties are expressed in another equivalent way, that is the following:

Property 1: ***Local accuracy***
When approximating the original model f for a specific input x, local accuracy requires the explanation model to at least match the output of f for the simplified input x.

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

The explanation model $g(x')$ matches the original model $f(x)$ when $x = h_x(x')$.
This clearly is the efficiency property. In order to see why it is sufficient to set $\phi_0 = E_X(f(X))$ and all $x'_i = 1$.

$$f(x) = \phi_0 + \sum_{i=1}^{M} \phi_i x'_i = E_X(f(X)) + \sum_{i=1}^{M} \phi_i$$

Property 2: ***Missingness***

If the simplified inputs represent feature presence, then missingness requires features missing in the original input to have no impact.

$$x'_i = 0 \Rightarrow \phi_i = 0$$

Missingness constrains features where $x'_i = 0$ to have no attributed impact.
This is the only property that is not among the properties of the "classic" Shapley values.

Property 3: ***Consistency***

If a model changes so that some simplified input's contribution increases or stays the same regardless of the other inputs, that input's attribution should not decrease.

Let $f_x(z') = f(h_x(z'))$ and $z' \setminus i$ denote setting $z'_i = 0$. For any two models f and f', if

$$f'_x(z') - f'_x(z' \setminus i) \geq f_x(z') - f_x(z' \setminus i)$$

for all inputs $z' \in \{0,1\}^M$, then $\phi_i(f', x) \geq \phi_i(f, x)$

From Consistency the Shapley properties of Linearity, Dummy and Symmetry follow as described in the appendix of [3].

The link between LIME and SHAP

Before we see how SHAP implements the computation of Shapley values it is interesting tho highlight the connection between LIME and SHAP. If we consider the family of interpretable models G of LIME constituted only by linear models, then the explanation model that LIME use is exactly given by equation 3 and thus LIME is an additive feature attribution method. LIME refers to simplified inputs x' as "interpretable inputs," and the mapping $x = h(x')$ converts a binary vector of interpretable inputs into the original input space. To find ϕ, LIME minimizes 1.
Here we find one of the main drawback of LIME that SHAP tries to overcome: while methods like LIME assume linear behavior of the machine learning model locally, and there is no theory on why this should work properly, Shapley values instead guarantees an explanation method with a solid theory, with the axioms of efficiency, symmetry, dummy, additivity, that give to the explanation a reasonable foundation.

Kernel SHAP

As we already mentioned SHAP is an alternative, kernel-based estimation approach for Shapley values inspired by local surrogate models. The main idea of SHAP is to use LIME, which we have seen that under some hypothesis can be seen as an additive feature attribution method, to recover the Shapley values, that is the only method with the three desirable properties (local accuracy, missingness and consistency) in the class of the additive feature attribution methods [1]. In order to fulfill this, the loss function L, the weighting kernel $\pi_{x'}$ and the regularization term Ω cannot be chosen in the classic heuristic way. To this end in [3] is reported the following theorem which specify the parameters to use in LIME in order to find the Shapley values.

Sampling

Since we are using LIME we need a dataset Z over which train the surrogate linear model. We do that by random sampling coalitions $z'_k \in \{0,1\}^M$, $k \in \{1,...,K\}$, where 1 means that the feature is present in the coalition and 0 that is absent. This K sampled coalitions become the dataset for the regression model. The target for the regression model is the prediction for a coalition. Before to move on, we have to translate coalitions of feature values to valid data instances (since we need the prediction of the coalitions), i.e. we need a function $h_x(z') = z$ where $h_x : \{0,1\}^M \to \mathbf{R}^p$. The function h_x maps 1's to the corresponding value from the instance x that we want to explain and the 0's by feature values of another instance that we sample from the data. In figure 3 it is shown a brief example of this sampling process in the contest of tabular data, taken by [4].

Theorem 1 *Under definition 3, the specific forms of $\pi_{x'}$, L and Ω that make solutions of equation 1 consistent with properties of local accuracy, missingness and consistency are:*

$$\Omega(g) = 0$$

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

$$L(f,g,\pi_{x'}) = \sum_{z' \in Z} [f(h_x^{-1}(z')) - g(z')]^2 \pi_{x'}(z')$$

Where Z is the training data and $|z'|$ is the number of non-zero elements in z'.

Since $g(z')$ in Theorem 1 is assumed to follow a linear form, and L is a squared loss, Equation 1 can still be solved using linear regression. As a consequence, the Shapley values from game theory can

[1] A proof of this statement can be found in the appendix of [3]

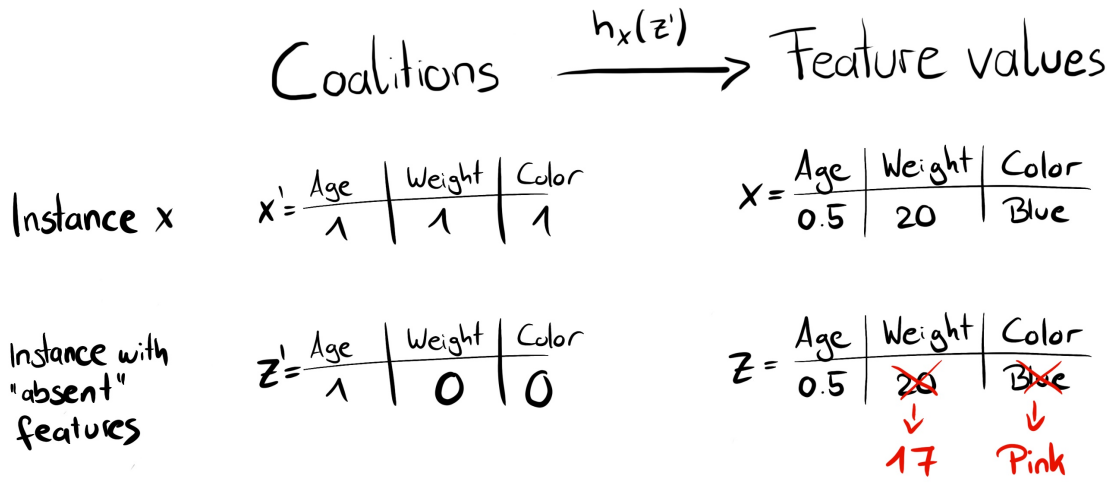

Figure 3: Function h_x maps a coalition to a valid instance. For present features (1), $h_x$maps to the feature values of x. For absent features (0), h_x maps to the values of a randomly sampled data instance

be computed using weighted linear regression.

The proof of Theorem 1 is shown in the appendix of [3]. Here we focus more on giving some intuition about this theorem.

Intuition

By the values given by the theorem we can clearly see that the main difference with LIME lay in how the instances are weighting. LIME weights the instances according to how close they are to the original instance. The more 0's in the coalition vector (i.e. the more values "replaced" and thus the more distance to the original point x), the smaller the weight in LIME. SHAP weights the sampled instances according to the weight the coalition would get in the Shapley value estimation. Small coalitions (few 1's) and large coalitions (i.e. many 1's) get the largest weights. The intuition behind it is: We learn most about individual features if we can study their effects in isolation. If a coalition consists of a single feature, we can learn about this feature's isolated main effect on the prediction. If a coalition consists of all but one feature, we can learn about this feature's total effect (main effect plus feature interactions). If a coalition consists of half the features, we learn little about an individual feature's contribution, as there are many possible coalitions with half of the features (maximal Entropy hence minimally informative).

3.3 NAMs

What we need before NAMs

Neural Additive Models (NAMs) belong to a model family called Generalized Additive Models (GAMs) [5]. In order to have a better understand of this method is then necessary an insight of this broader model family calls GAMs which are also an important and still hot topic of XAI. For example the state-of-the-art in high-accuracy, interpretable generalized additive models, that are GAM [6] and GA^{2}M [7] based on regularized boosted decision trees have been successfully applied to an healthcare datasets [8].
As already did in this work, in order to keep this work self-consistent, we will start first by briefly introduce linear regression models and their generalization to GAMs passing through GLMs (Generalized Linear Models).

From linear regression to GAMs

A linear regression model is simply a model of this kind:

$$y = \beta_0 + \beta_1 x_1 + ... + \beta_K x_K$$

Here, the prediction is modeled as a weighted sum of the features. This is maybe both the main advantage and disadvantage of this kind of model. Indeed thanks to the linearity the estimation procedure is simple and moreover these linear equations have an easy to understand interpretation on a modular level. However everything have a cost, the facility of the model is paid in terms of its expressivity, indeed the assumptions made for this model are often violated in reality: the features might interact and the relationship between the features and the outcome might be nonlinear.

A first step toward a more complex model is the Generalized Linear Models. GLMs allow for response distributions other than normal, and for a degree of non-linearity in the model structure [9]. Mathematically GLMs have the following structure:

$$g(E_Y(y|x)) = \beta_0 + \beta_1 x_1 + ... + \beta_K x_K$$

As we can see GLMs are composed by three components: the link function g, the weighted sum $X^T\beta$ and a probability distribution from the exponential family that defines E_Y. Thus, thr Basic model formulation is much the same as for linear models, except that a link function and distribution must be chosen. Of course, if the identity function is chosen as the link, along with the normal distribution, then ordinary linear models are recovered as a special case. [9]

Now, we will take a further step toward a more generalized model by simply allow that each feature contributes to the outcome no more in a linear way $\beta_j x_j$ but as the output of a generic arbitrary function $f(x_j)$. Mathematically,in GAMs we have the following relation:

$$g(E_Y(y|x)) = \beta + f(x_1) + f(x_2) + ... + f(x_K) \tag{4}$$

With this formulation we allow the model to learn nonlinear relationships. It is important to highlight that this more flexibility of our model comes at the cost of interpretability. Any link function (in a GLM) that is not the identity function complicates the interpretation; nonlinear feature effects are either less intuitive. Here we don't explain how to learn nonlinear functions in GAMs, because for our purpose is not useful (i.e. to describe NAMs), however the interested reader can see [9] to learn more about GAMs and how their nonlinear functions are learnt,

Introduction to NAMs

At this point we have all the basic ingredients to introduce the main topic of this section: Neural Additive Models (NAMs). NAMs are a particular restriction on the structure of neural networks, which yields to a family of glass-box models, that are inherently interpretable. Methodologically, NAMs belong to a model family called Generalized Additive Models (GAMs).

Recalling 4, NAMs are essentially GAMs in which the function f_i in 4 is parametrized by a neural network.

In other words, NAMs learn a linear combination of networks that each attend to a single input feature: each f_i in 4 is parametrized by a neural network. These networks are trained jointly using backpropagation and can learn arbitrarily complex shape functions. Interpreting NAMs is easy as the impact of a feature on the prediction does not rely on the other features and can be understood by visualizing its corresponding shape function (e.g., plotting fi(xi) vs. xi). [10]

It is worth to remark that the graphs learned by NAMs are an exact description of how NAMs compute a prediction since the model is inherently interpretable.

Why NAMs instead of GAMs?

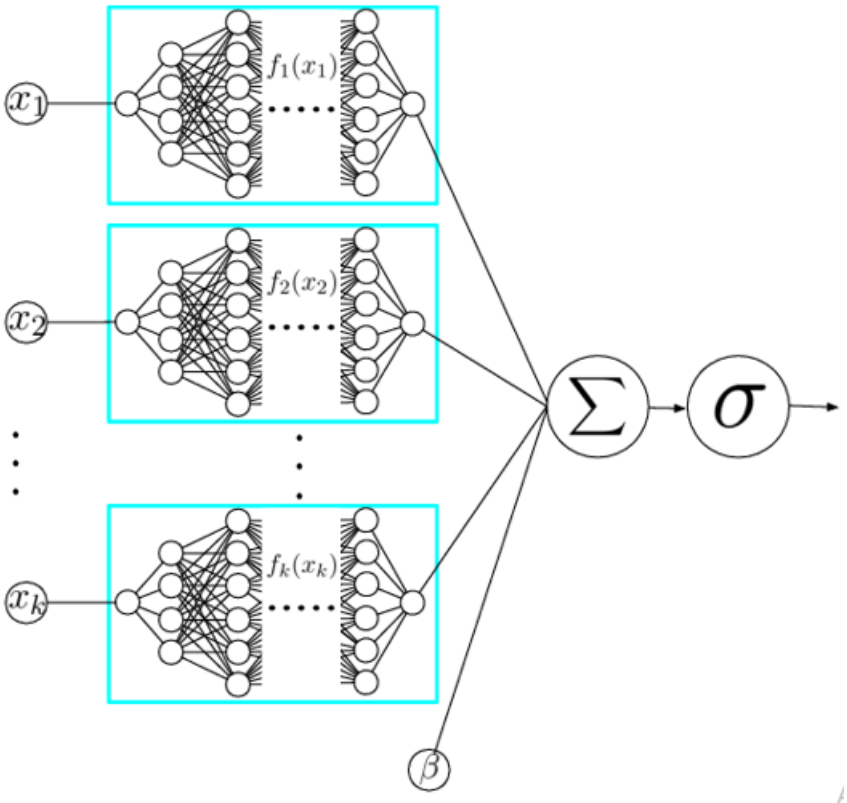

Figure 4: NAM architecture for binary classification. Each input variable is handled by a different neural network.

At this point one can be ask what benefits provide the use of NAMs instead of the classical GAMs, the authors of NAMs in [10] illustrate the whole advantages that one might have exploiting NAMs instead of GAMs. Here we report only a few of them, in order to maintain compactness in this work, the interested reader can find all the motivations provided by the authors in [10].

- NAMs, due to the flexibility of NNs, can be easily extended to various settings problematic for boosted decision trees. For example, extending boosted tree GAMs to multitask, multiclass or multi-label learning requires significant changes to how trees are trained, but is easily accomplished with NAMs without requiring changes to how neural nets are trained due to their composability (INSERT SECTION OF MULTITASK NAMs). Furthermore, the differentiability of NAMs allows them to handle problems where differentiability in the model is required.
- Graphs learned by NAMs are not just an explanation but an exact description of how NAMs compute a prediction. As such, a decision-maker can easily interpret NAMs and understand exactly how they make decisions. This would help harness the expressivity of neural nets on high-stakes domains with intelligibility requirements
- NAMs are more scalable as inference and training can be done on GPUs/TPUs or other specialized hardware using the same toolkits developed for deep learning over the past decade – GAMs currently cannot.

Intelligibility of NAMs

The intelligibility of NAMs results in part from the ease with which they can be visualized. SInce each feature is handled independently by a learned shape function parameterized by a neural net, one can get a full view of the model by simply graphing the individual shape functions.

Brief description of the performance of NAMs

As reported in [10] NAMs are competitive in accuracy to GAMs and accurate alternatives to prevalent interpretable models (e.g., shallow trees) while being more easily extendable than existing GAMs due to their differentiability and composability.

Multitask NAM

In this part we will discuss the precedent cited composability of NAMs, in particular we will see how, thanks to this property, NAMs can be easily extended in the case of multitask learning (MTL) [11]. Notice that this extension is not an easy task in any EBMs (Energy Based Models) or in any major boosted-tree package [10]. In NAMs we can exploit the composability of neural networks in order to train multiple subnets for each feature.

In particular the multitask NAM architecture is identical to that of single task NAMs except that each feature is associated with multiple subnets and the model jointly learns a task-specific weighted sum over their outputs (of the subnets corresponding to a single feature) that determines the shape function for each feature and task.

Then, the outputs corresponding to each task are summed and a bias is added to obtain the final prediction score.

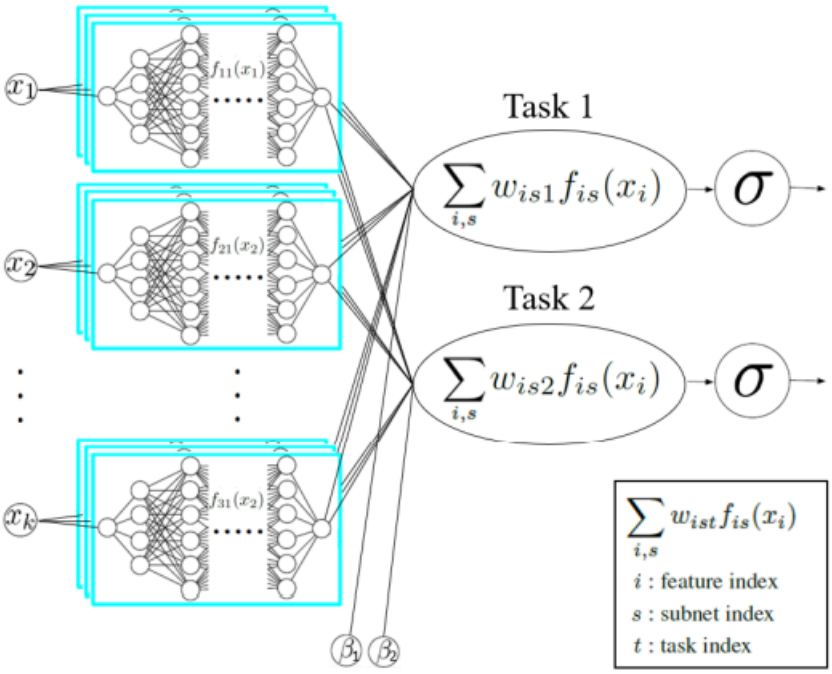

Figure 5: **Multitask NAM architecture** for binary classification. Multiple subnets are trained on each input feature and weighted sums are learned over the subnets.

Doing in this way the model can learn task-specific weights over these subnets to allow sharing of subnets (shape functions) across tasks.

Figure 5 shows a multitask NAM architecture that can jointly learn different feature representations for each task. It is worth highlighting that this extension of NAMs preserves its intelligibility and modularity.

In [10] is shown an interesting example of the application of a multitask NAMs applied on a synthetic dataset that can provide a useful insight on this extension.

Advanced to NAMs: only s improving the expressivity of NAMs by incorporating higher-order feature interactions citing the linear case

Finally, we conclude this section about NAMs by showing a simple advancement of this method. We can indeed improve the expressivity of NAMs by incorporating higher-order feature interactions, just as it is already done in GAMs. Namely the mathematical formulation with these higher-order feature interactions becomes as following:

$$g(E_Y(y|x)) = \beta + f(x_1) + f(x_2) + ... + f(x_K) + f(x_{i1}, x_{j1}) + ... + f(x_{iP}, x_{jP})$$

While such interactions may result in more expressivity, they might worsen the intelligibility of the learned NAM. Thus, finding a small number of crucial interactions seems important for more expressive yet intelligible NAMs.

Other further interesting advancements of NAMs can be found in [10].

4 Metrics for explainable AI

Introduction

While numerous explanation methods have been developed so far, there is still a need for evaluation metrics to quantify their quality to determine whether and to what extent the offered explainability achieves the defined objective. These permit also to compare available explanation methods and suggest the best explanation from the comparison over a specific task. As pointed out in [13] this field is still immature. For this reason, in the following sections we avoided to go deep on some specific metrics and we opted for a more general summary of the subject in order to give to the reader a more general viewpoint that can be exploited in the near future when the research on this field will be mature. Before moving on, it is useful to illustrate why there is a necessity for evaluation methods. This is done by Markus et al. [15], in particular they divided the goal of evaluation into two-fold:

- To compare available explanation methods and find preferred explanations from the comparison. The challenge is that there is no ground truth when evaluating post-hoc explanations, as the real inner workings of the model is not known [16].
- To assess if explainability is achieved in an application. The focus of the assessment lies in determining if the provided explainability achieves the defined objective [17].

4.1 Taxonomy of Interpretability Evaluation

In this section we will present a widely cited taxonomy, probably the most, that is the one by Doshi-Velez and Kim [12]. This one divides the evaluation approaches for interpretability into three categories: application-grounded, human-grounded, and functionally-grounded (Figure 6).

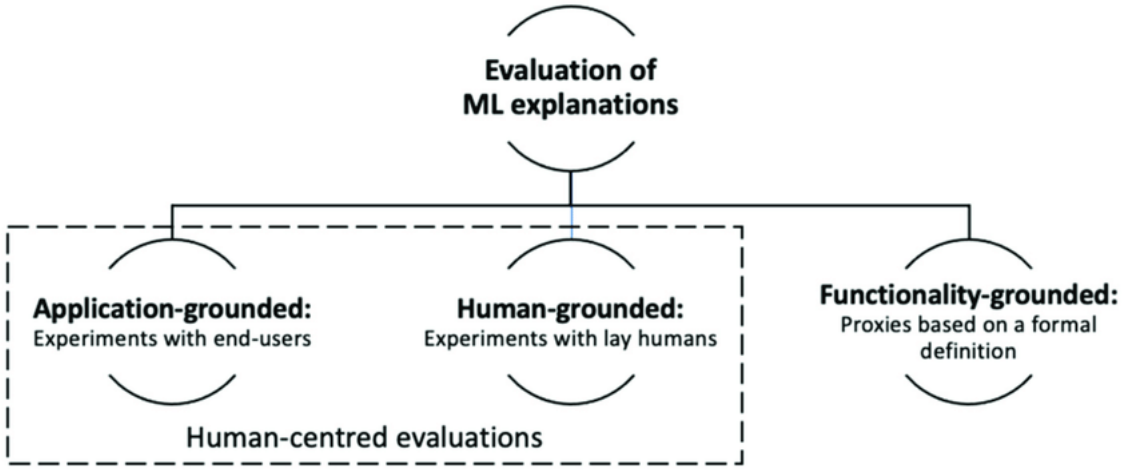

Figure 6: Taxonomy of evaluation of machine learning explanations.

It is worth highlighting that while human evaluation is essential to assessing interpretability, human-subject evaluation is not an easy task. Moreover, human experiment needs to be well-designed to minimize confounding factors, consumed time, and other resources.

4.1.1 Application-grounded Evaluation

Application-grounded evaluation involves conducting human experiments within a real application. It directly tests the objective that the system is built for in a real-world application, and, thus, performance with respect to that objective gives strong evidence of the success of explanations. Examples of experiments include:

- Domain expert experiment with the exact application task.
- Domain expert experiment with a simpler or partial task to shorten experiment time and increase the pool of potentially-willing subjects.

In both cases, an important baseline is how well human-produced explanations assist in other humans trying to complete the task

4.1.2 Human-grounded Evaluation

Human-grounded evaluation is about conducting simpler human-subject experiments that maintain the essence of the target application. Differently to application-grounded evaluation evaluations can be carried out with lay humans instead of domain experts. This entails some advantages such as a bigger subject pool and less expenses for the evaluation (since we do not have to compensate highly trained domain experts). Ideally, this evaluation approach will depend only on the quality of the explanation, regardless of the types of explanations and the accuracy of the associated prediction. Examples of potential experiments include:

- Binary forced choice: humans are presented with pairs of explanations, and must choose the one that they find of higher quality.
- Forward simulation/prediction: humans are presented with an explanation and an input, and must correctly simulate the model's output (regardless of the true output).
- Counterfactual simulation: humans are presented with an explanation, an input, and an output, and are asked what must be changed to change the method's prediction to a desired output.

4.1.3 Functionality-grounded Evaluation

Functionally-grounded evaluation requires no human experiments; instead, it uses some formal definition of interpretability as a proxy for explanation quality. The absence of the human-in-the-loop is what makes this evaluation so interesting. Indeed, in this way we overcame the general problem that human-subject experiments require time and costs both to perform and to get necessary approvals, which may be beyond the resources of a machine learning researcher. Moreover they can be applied also when a method is not yet mature or when human subject experiments are unethical. However, functionally-grounded evaluations are most appropriate once we have a class of models or regularizers that have already been validated, e.g. via human-grounded experiments.The main challenge here, of course, is to determine what proxies to use. Examples of experiments include:

- Show the improvement of prediction performance of a model that is already proven to be interpretable (assumes that someone has run human experiments to show that the model class is interpretable).
- Show that one's method performs better with respect to certain regularizers—for example, is more sparse, compared to other baselines (assumes someone has run human experiments to show that the regularizer is appropriate).

4.2 Metrics for human-centred evaluations

First of all, the quality evaluation of ML explanations is inherently a subjective concept. Therefore, human-centered evaluations are indispensable to the overall quality evaluations. A critical role for the correct evaluation of ML systems is played by the choice of evaluation metrics. In the context of human-centered evaluations, two types of evaluation metrics can be found in explainable ML research:

- **Subjective metrics.** Subjective questionnaires are designed for users on tasks and explanations, and are asked during or after task time to obtain user's subjective responses on tasks and explanations. Examples of subjective metrics are user trust, confidence, and preference, which have been largely embraced as the focal point for the evaluation of explainable systems.
- **Objective metrics.** This refers to objective information on tasks or humans before, after, or during the task time. Examples include human metrics, such as physiological and behavior indicators of humans, during ML-informed decision-making, or task related metrics, such as task time length and task performance. Other forms of objective metrics can be based for example on quantifying response times in user studies as well as agreement between human labels and model predictions.

As we can read in [14], despite that there are various investigations on human-centered evaluations, there are no agreed criteria on human-centered evaluations, especially human experimental design and subjective measures to be used, which make it hard to compare the quality of different evaluations. In order to find these aspects, [14] reports a series of open questions:

- What subjective measures should be used for an explanation evaluation?
- What are human's tasks in experiments?
- How many participants should be recruited and what are their background?

It can be reasonable argue that human-centered evaluations are dependent on application domains and target users. However, finding some common criteria in human-centered evaluations is helpful for effective evaluations as well as they are necessary in the comparison between methods based only on human-centered metrics.

It is understandable that human-centered evaluations are dependent on application domains and target users. However, from a practical perspective, we argue that the criteria on common components in human-centered evaluations are helpful for effective evaluations. Therefore, the future work of the human-centered evaluations needs to focus on the investigation of effective user experiment designs as well as innovative approaches used to collect subjective measures for explanation evaluations.

Metrics for Functionality-Grounded evaluations

As said in [14], a possible way to objectively evaluate explanations is to verify whether the explanation mechanism satisfies (or does not satisfy) certain axioms, or properties. Therefore, various quantitative metrics are developed to objectively assess the quality of the explanation, according to these evaluation properties. Before moving on, it is then important to first describe these properties.

Brief description of properties

According to [15] explainability is composed of two main parts, namely interpretability and fidelity (sometimes named faithfulness). Follow the work done in [15] and [18], interpretability can have the following properties:

- Clarity implies that the explanation is unambiguous.
- Parsimony means that the explanation is presented in a simple and compact form.
- Broadness, which describes how generally applicable is an explanation.

For what regard fidelity we have the following properties:

- Completeness means that the explanation describes the entire dynamic of the ML model.
- Soundness that concerns how correct and truthful the explanation is.

In figure 7 is reported the concept of explainability and its related properties.

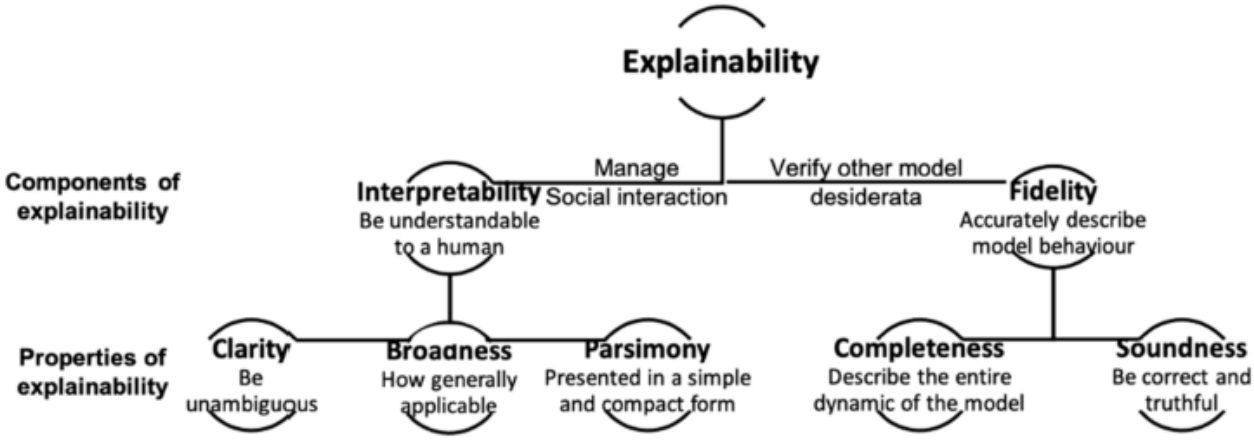

Figure 7: Definition of machine learning (ML) explainability and related properties (from [15]).

The new taxonomy on ML methods

Following the work of [14], we will divided the ML explanation methods into three types, as done in [15]:

- Model-based explanations. It refers to explanations that use a model to explain original task models. In this category, either the task model itself (e.g., decision tree) is used as an explanation or more interpretable models are generated to explain the task model.
- Attribution-based explanations. This kind of explanation ranks or measures the explanatory power of input features and use this to explain the task model.
- Example-based explanations. This kind of method explains the task model by selecting instances from the training/testing dataset or creating new instances. For example, selecting instances that are well predicted or not well predicted by the model as explanations, or creating counterfactual examples as explanations.

Notice that this new taxonomy is not perfect, for example both LIME and NAMs fall into two categories, namely the model-based and the attribution-based explanations. Reasonably, one might ask why use this new taxonomy here instead of the previous one. The answer is because the functionality-grounded metrics are better categorized following this division. In what follows we will illustrate very briefly some quantitative metrics for measuring the quality of three types of explanation methods. The interested reader can find out more about these metrics as well as their origin in [14].

Explanation Types	Quantitative Metrics	Properties of Explainability				
		Interpretability			Fidelity	
		Clarity	Broadness	Parsimony/ Simplicity	Completeness	Soundness
Model-based explanations	Model size			✓		
	Runtime operation counts			✓		
	Interaction strength			✓		
	Main effect complexity			✓		
	Level of (dis)agreement	✓				✓
Attribution- based explanations	Monotonicity					✓
	Non-sensitivity Sensitivity					✓
	Effective complexity		✓	✓		
	Remove and retrain					✓
	Recall of important features					✓
	Implementation invariance					✓
	Selectivity					✓
	Continuity	✓				
	Sensitivity-n					✓
	Mutual information		✓	✓		✓
Example-based explanations	Non-representativeness			✓	✓	
	Diversity			✓		

Figure 8: Quantitative metrics for machine learning (ML) explanations.

5 Conclusion

In this review we have seen the necessity of explanation methods in the context of machine learning. Blindly trusting the results of an highly predictive classifier is, by today's standard, inadvisable, due to the strong influence of data bias, trustability, and adversarial examples [13].
After we have reported a general and widely used taxonomy of XAI we have opted for a different approach than the classic one used for survey on this field. Indeed, due to the several time constraints to complete this work, an exhaustive survey is almost impossible (at least for beginners on this subject). Therefore we decided to focus only on a small set of significant algorithms but on the contrary of classic surveys we went deeply inside of them. Since the small set, we had to choose carefully which methods to describe.
The three methods that we illustrated are respectively LIME, SHAP and NAMs. The reason beyond the choice of the first two methods are mainly because they are the most cited methods among all the XAI algorithms. This has the advantage that the most state-of-the-art of XAI methods take knowledge for granted about these "classic" algorithms and moreover they are often used as baseline for the performances of competing methods. Furthermore, since SHAP is based on Shapley Values, it has also a solid theoretical foundation in game theory [4], which is essential if we want to guarantee some axioms. Finally the choice of NAMs is motivated mainly by two factors. The first one is that we wanted to show also some state-of-the-art approaches and moreover this method has forced us to cover, even if at very high level, another important SoTA algorithm currently used, with some of its implementations, namely GAMs. The second reason is simply a subjective preference based on the proximity of this method with our background.
In the last part we discussed about XAI evaluation metrics. As already exposed, this is still an immature field in rapid evolution, furthermore this has motivated us in giving a more general perspective about the topic. With each step toward the whitening of these black-box AI algorithms, we are one step closer to decode AI solutions that surpass human ability while also removing some of the last trust barriers which stops many peoples and industries in fully embrace this new technologies.

References

[1] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why Should I Trust You?"," in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16. New York, New York, USA: ACM Press, 2016, pp. 1135–1144.

[2] AI Explainability Whitepaper - Google

[3] S. M. Lundberg and S. I. Lee, "A unified approach to interpreting model predictions," in Advances in Neural Information Processing Systems, 2017, pp. 4765–4774.

[4] Molnar, Christoph. "Interpretable machine learning. A Guide for Making Black Box Models Explainable", 2019. https://christophm.github.io/interpretable-ml-book/.

[5] Trevor Hastie and Robert Tibshirani. Generalized Additive Models. Chapman and Hall/CRC, 1990.

[6] Yin Lou, Rich Caruana, and Johannes Gehrke. Intelligible models for classification and regression. SIGKDD, 2012.

[7] Yin Lou, Rich Caruana, Johannes Gehrke, and Giles Hooker. Accurate intelligible models with pairwise interactions. SIGKDD, 2013.

[8] Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. SIGKDD, 2015

[9] S. Wood. Generalized additive models: an introduction with R. CRC Press, 2006

[10] R. Agarwal, N. Frosst, X. Zhang, R. Caruana, and G. E. Hinton, "Neural additive models: Interpretable machine learning with neural nets," arXiv preprint arXiv:2004.13912, 2020.

[11] Rich Caruana. Multitask learning. Machine Learning, 1997.

[12] Doshi-Velez, F.; Kim, B. Towards A Rigorous Science of Interpretable Machine Learning. arXiv 2017, arXiv:1702.08608.

[13] Das, A.; Rad, P. Opportunities and challenges in explainable artificial intelligence (xai): A survey. arXiv 2020, arXiv:2006.11371.

[14] Jianlong Zhou, Amir H. Gandomi, Fang Chen, and Andreas Holzinger. Evaluating the Quality of Machine Learning Explanations: A Survey on Methods and Metrics. Electronics, 10(5):593, January 2021.

[15] Markus, A.F.; Kors, J.A.; Rijnbeek, P.R. The Role of Explainability in Creating Trustworthy Artificial Intelligence for Health Care: A Comprehensive Survey of the Terminology, Design Choices, and Evaluation Strategies. arXiv 2020, arXiv:2007.15911.

[16] Samek,W.; Montavon, G.; Vedaldi, A.; Hansen, L.K.; Müller, K.-R. (Eds.) Explainable AI: Interpreting, Explaining and Visualizing Deep Learning; Lecture Notes in Artificial Intelligence, Lect.Notes ComputerState-of-the-Art Surveys; Springer: Berlin/Heidelberg, Germany, 2019; ISBN 978-3-030-28953-9.

[17] Lipton, Z.C. The Mythos of Model Interpretability: In Machine Learning, the Concept of Interpretability Is Both Important and Slippery. Queue 2018, 16, 31–57.

[18] Lombrozo, T. Explanatory Preferences Shape Learning and Inference. Trends Cognit. Sci. 2016, 20, 748–759.

[19] A. Adadi and M. Berrada, "Peeking inside the black-box: A survey on explainable artificial intelligence (XAI)," IEEE Access, vol. 6, pp. 52138–52160, 2018.