Towards a Generic Framework for Black-box Explanation Methods (Extended Version)

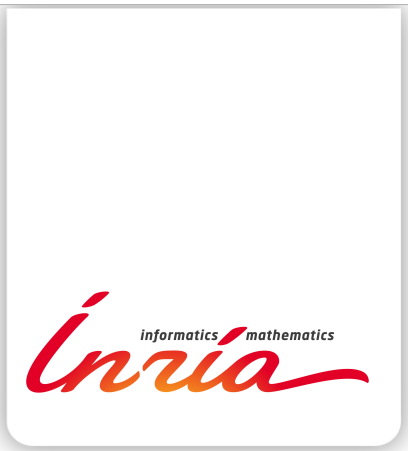
Clement Henin, Daniel Le Métayer

Towards a Generic Framework for Black-box Explanations Methods (Extended Version)

Clément Henin, Daniel Le Métayer

RESEARCH REPORT
N° 9276
May 2019
Project-Team PRIVATICS

ISSN 0249-6399 ISRN INRIA/RR--9276--FR+ENG



Towards a Generic Framework for Black-box Explanations Methods (Extended Version)

Clément Henin, Daniel Le Métayer

Project-Team PRIVATICS

Research Report n° 9276 — May 2019 — 28 pages

Abstract: The main goal of this research report is to define a generic framework for black-box explanation methods in order to make it easier to compare and classify different approaches. We focus on two components of this framework, called respectively "Sampling" and "Generation", which are characterized formally and used to build a taxonomy of explanation methods. We also describe precisely how each method can be expressed in the framework.

Key-words: Algorithmic decision system, explanability, explanation, transparency, black-box model, machine-learning, artificial intelligence

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Vers un cadre générique pour l'explication en mode boîte noire des systèmes de décision algorithmique (version étendue)

Résumé : Dans ce rapport de recherche, nous proposons un cadre générique pour décrire les méthodes d'explication fonctionnant en mode "boîte noire", l'objectif étant de faciliter la comparaison et la classification de ces méthodes. Nous définissons formellement deux composantes principales, appelées respectivement l'"Echantillonnage" et la "Génération", qui sont ensuite utilisées pour construire une taxonomie des méthodes d'explication. Nous décrivons aussi précisément la manière dont les méthodes de la littérature s'expriment dans ce cadre.

Mots-clés : Système de décision automatique, explicabilité, explications, transparence des algorithmes, modèle boîte-noire, apprentissage, intelligence artificielle

Contents

1 Motivations

Algorithmic Decision Systems (hereafter "ADS") are increasingly used in many areas, sometimes with a major impact on the lives of the people affected by the decisions. Some of these systems make automatic decisions, for example to reduce or to increase the speed of an autonomous car, while others only make suggestions that a human user is free to follow or not. In some cases, the user is a professional, for example a medical practitioner or a judge, while in other cases he is an individual, for example an internet user or a consumer. Some ADS rely on traditional algorithms, while others are based on machine learning (hereafter "ML") and involve complex models such as neural networks, support vector machine or random forest. Regardless of these considerations, when an ADS can have a significant impact, its design and validation should ensure a high level of confidence that it complies with its requirements.

Explainability has generated increased interest during the last decade because the most accurate ML techniques often lead to opaque ADS and opacity is a major source of mistrust. Indeed, even if explanations are not necessarily a panacea, they can play a key role, not only to enhance trust in the system, but also to allow its users to better understand its outputs and therefore to make a better use of it. Explanations can take different forms, they can target different types of users (hereafter "explainees") and different types of methods can be used to produce them. In this research report, we focus on a category of methods, called "black-box", that do not make any assumption of the availability of the code of the ADS or its implementation techniques. The only assumption is that input data can be provided to the ADS and its output data can be observed.

Explainability is a fast growing research area and many papers have been published on this topic during the last years. These papers define methods to produce different types of explanations in different ways but they also share a number of features. The main goal of this research report is to bring to light a common structure for Black-box Explanation Methods (BEM) and to define a generic framework allowing us to compare and classify different approaches. This framework consists of three components, called respectively *Sampling*, *Generation* and *Interaction*. The need to conceive an explanation as an interactive process rather than a static object has been argued in a very compelling way by several authors [17, 18, 19]. It must be acknowledged, however, that many contributions in the XAI community do not emphasize this aspect. Therefore, in view of space limitation, we do not discuss the precise form that explanations and interactions with the explainee can take, and focus on the *Sampling* and *Generation* components. We characterize these components formally and use them to build a taxonomy of explanation methods. We come back to the link with the *Interaction* component in the conclusion. Beyond its interest as a systematic presentation of the state of the art, we believe that this framework can also provide new insights for the design of new explanation systems. For example, it may suggest new combinations of *Sampling* and *Generation* components or criteria to choose the most appropriate combination to produce a given type of explanation.

We first provide some intuition about the framework and describe it formally in Section 2. Then we present in Section 3 a taxonomy of black-box explanation systems derived from our framework and describe the instantiation of the framework to an example of BEM. We illustrate the interest of the framework for the design of explanation systems in Section 4. Finally, we provide an overview of related work in Section 5 and conclude with perspectives in Section 6.

2 Description of the framework

We first provide an overview and some intuition about our framework in Section 2.1, before presenting the *Sampling* and *Generation* components more formally in Section 2.2 and Section

2.3 respectively.

2.1 Overview

To introduce our framework, we consider the concrete example of a spam classifier. The system takes as input the text of an email and outputs the probability of this email being a spam. Ideally, an explanation system (herefater, "explainer") should be able to answer a wide range of questions because different explainees have different interests, motivations and levels of expertise. For example, a *user of the spam classifier* may want to ask questions to better understand the system or its behavior in specific circumstances. Possible questions on his part include "Did the signature part of email x_e have an impact on the fact that it has been classified as a spam?" or "Why is email x_e classified as a spam and not email y_e?" The explanations can be useful to enhance his trust in the classifier or to allow him to understand how to modify its parameters if it does not behave as expected. On the other hand, *the designer of the system* may have more precise requests such as are "What are the main features used by the classifier to decide that an email is likely to be a spam and what are their respective weights?" Since we assume that the code of the classifier is not available, the explainer can only build emails, submit them to the classifier and analyze the results. For example, to answer the first question of the user, the explainer can create different versions of x_e with and without the signature part, or with different pieces of text in the signature part. The explainer has then to compute the answer based on the results of the classifier and to present it to the explainee.

This simple example highlights the three main tasks of an explainer which are pictured in Figure 1 : (i) the *Interaction* task, which includes the analysis of the questions of the explainee and the presentation of the explanations in an intelligible way; (ii) the selection of inputs to submit to the system to be explained, which is called the *Sampling* task; and (iii) the analysis of the links between the selected inputs and the corresponding outputs of the system to generate the content of the explanations, which is called the *Generation* task. In many cases, the *Sampling* task and the *Generation* task are applied sequentially but it is sometimes useful to apply them iteratively, to be able to adjust the set of samples to the needs of the *Generation* task. As stated in the introduction, the *Interaction* task has not received as much attention as the two other tasks in the literature so far. Therefore, for the sake of conciseness we focus on the *Sampling* and the *Generation* tasks in this research report. We propose formal characterizations of these tasks which are generic enough to encompass existing proposals and to compare them on a rigorous basis, as discussed in Section 3 and sketched in Table 1.

Name	Description	Example
M	Black-box model	The spam classifier
I	Input space of M	Space of emails
O	Output space of M	$[0, 1]$
E	Scope of the explanation	Email x_e
S	Samples (product of the sampling step)	Emails with changed signature
D	Dataset describing the overall population	Training set of M

2.2 Sampling

The role of the *Sampling* task is to select appropriate inputs (or "samples") to answer a question of the explainee about a model M. The choice of the samples may depend on a number of

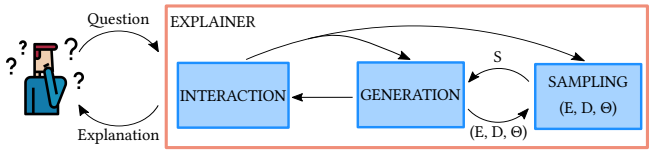


Figure 1: The three main tasks of the explainer

factors. The first aspect to take into consideration is whether the question concerns the whole model or specific inputs. We call E the scope of the explanation. If the question concerns a single input x_e, then $E = \{x_e\}$; if the question is about the whole model M, then $E = D$ with D a representation of the population (possible inputs to M) available to the explainer[1]. In general, E and D could be any subset of possible input values. We call I this set of input values, which can be seen as the support set (or type set) of D. In the spam filter example, I is the set of all possible emails (i.e. the set of all texts of a given format) and D represents the actual data set of emails available to the explainer, which is used, for example, to estimate the distributions of the features. In some cases, the explainer does not have any information about this distribution, which is denoted by $D = \emptyset$. The result of the *Sampling* task is a set of samples $S = \{x_1, ..., x_n\} \in I^n$. Different values of D may give rise to different sampling strategies. For example, to address the first question of the user of the spam filter about the impact of the signature on the classification of x_e, a possible option is to select a single sample obtained by removing the signature part of x_e. This strategy does not require any information about the actual distribution of the population and can therefore be applied with $D = \emptyset$. However, the answer might not be realistic or precise enough. A more elaborate strategy would be to replace the original signature of x_e by real signatures obtained from many other emails. This strategy requires information about the actual distribution of the population ($D \neq \emptyset$) in order to ensure that the sample set reflects the reality. We can now define the sampling procedure as follows [2]:

$$S = \{h_\theta(x_e, x_p) \mid (\theta, x_e, x_p) \in \Theta \times E \times D, F(\theta, x_e, x_p) = 1\} \tag{1}$$

with

$$h_\theta : E \times D \to I \tag{2}$$

Θ is the set of parameters for the sampling and F is a filter function. In a nutshell, the θ parameter makes it possible to generate several samples for each pair (x_e, x_p) while F makes it possible to generate samples only for a selection of pairs (x_e, x_p). In our spam filter example, E is limited to a single email to be explained ($E = \{x_e\}$), there is no need for parametrization so we take $\Theta = \{0\}$ and we assume that D contains 1000 emails. Function $h_0(x_e, x_p)$ returns an email sample obtained from x_e by replacing its signature by the signature of x_p. If F is the constant function that always returns 1 ($F(\theta, x_e, x_p) = 1$), then the sampling procedure generates 1000 perturbed version of x_e with signatures extracted from the emails in D. Another option could be to use a filter function F relying on a notion of distance and selecting only emails close to x_e, or a function F selecting only emails with the same subject part as x_e. The θ parameter can be used to customize the sampling function. For instance, if both the header and the signature of the email are taken into consideration, θ could specify which part of the email is replaced (header, signature or both).

[1] It should be noted that D is actually a multiset since it can involve multiple occurrences of the same value to reflect the distribution of the values in the real population.

[2] If Θ, D or E is empty, it should be replaced in (1) by 0, otherwise the product space would also be empty.

2.3 Generation

When the set S of samples is available, the next step consists in providing its elements as inputs to the model M and to collect the outputs. The set $\{ (x, M(x)) \mid x \in S\}$ is the raw materiel to build the explanations. Even if explanations can take many different forms, the *Generation* task can conceptually be split into two parts: the computation of a *proxy* of the model M and the construction of an explanation based on this proxy. We call the former *model generation* (G_M) and the latter *explanation generation* (G_E). In some cases, the proxy model is considered as the explanation itself, in which case no explanation generation is necessary; in other cases, the proxy model is an intermediate step to derive the explanation delivered to the explainee. We emphasize that we present a conceptual view of the *Generation* task here: the actual implementation of a BEM does not necessarily involve the construction of the proxy model. Coming back to the spam classifier example, an option for the *Generation* task is to train a simple rule-based model on the samples to predict the output of the classifier. An example of rule generated by this step could be: "If the signature of the email is less than 60 characters long, then the classifier will consider that it is a spam; otherwise it will be considered as an acceptable email". Because such rules are easily interpretable, they can directly be used as explanations. In other situations, either because the type of model used is too complex or the model is too big to be understandable (for example if it involves a large number of rules), simpler explanations have to be generated from the proxy model. This explanation generation phase can return, for example, the most important feature(s) of the input. For the spam classifier, the answer in this case could be: "The length of the signature part and the number of typos are the two most important features used by the system to decide if an email is a spam".

Technically speaking, the proxy model is denoted by F_w, which is a function of the same type as the model M, parameterized by w:

$$F_w : I \to O \tag{3}$$

The core of the *Generation* task is to find the best F_w to answer the question of the explainee, which amounts to find the optimal values of w. Optimality can be defined formally using sets of constraints $o_i(w, S) \in \mathbb{B}$ and criteria $c_i(w, S) \in \mathbb{R}$ where $\mathbb{R}$ and $\mathbb{B}$ are the sets of real numbers and booleans respectively. The global objective takes the following form:

$$\begin{aligned} w^* = \underset{w}{\operatorname{argmin}} \quad & \sum_i \lambda_i c_i(w, S) \\ \text{subject to} \quad & o_i(w, S) \end{aligned} \tag{4}$$

where $\lambda_i \in \mathbb{R}$ are used to weight the criteria. In many methods, the objective is to find the parameters w such that the proxy is as close as possible to M on the samples of S. However, using both criteria and constraints provides a great flexibility, which contributes to the generality of our framework. Finding a good explanation is often a question of compromise. A typical example is finding the right balance between precision and complexity – often used as a characterization of understandability. For example, a simple explanation of the spam classifier that would be accurate (i.e. predicting the actual result of the classifier) on only seventy percent of its inputs would not be acceptable; on the other hand, an accurate explanation that would take the form of several pages of rules would provide little insight to the user. As discussed in the following section, criteria and constraints can be used to define the priorities among objectives.

The second step of the of the *Generation* task, the explanation generation, is generally less technical. For instance, Shapley [29], PDP ICE [13] or VIN [12] compute sums of elements to produce a plot or specific numerical values; LIME [21] extracts the coefficients of a linear function; LEMNA [10] and Local-gradients [2] derive from F_w slopes in the neighborhood E.

3 Taxonomy of black-box explanation systems

We first show in Section 3.1 how our generic framework can be used to analyze and classify existing explanation methods (Table 1). In view of space limitations, we cannot provide the details of the instantiation of the framework for each method of Table 1 but we present as an illustration the case of counterfactual explanations in Section 3.2. The interested reader can find the details of the definitions in our framework of the methods of Table 1 in the appendices.

3.1 Key features

The generic definitions introduced in the previous section allow us to highlight the range of choices in the design of a BEM and to classify existing methods based on these choices. Table 1 summarizes these design choices and the types of explanations produced by these methods. In this section, we provide some intuition about Table 1, considering successively the design choices related to the *Sampling* task (columns to the left), the *Generation* task (middle columns) and general choices (two columns to the left).

Design choices related to the *Sampling* task:

- The set E makes it possible to express the focus of the explanation: if the explainee is interested in a specific input data (e.g. an email x_e) then E is a singleton set (e.g. $E = \{x_e\}$). At the other end of the spectrum, the explainee may be looking for a global explanation of the system, which is expressed by $E = D$. Ideally, the explainee should be able to choose any scope between these two extremes. However, as shown in Table 1, existing methods assume a fixed scope, which is almost always D or a singleton set. The only exception is QII [5] that makes it possible to focus on a group of input data.

- In general, a model may behave differently on different segments of the population. For instance, face recognition systems are more accurate for white males than for non-white females [3]. Therefore, taking into account or not the distribution of the population is an important feature of a BEM. It appears with the set D in our framework. D is the information available to the BEM about the population "*Pop.*". Table 1 shows that all methods except LIME and LEMNA take the population into account. The population does not have any impact on the explanations produced by LIME and LEMNA ($D = \emptyset$) because they build samples by random masking of the input data. The relevance of the use of the population depending on the question of the explainee is further discussed in Section 4.

- The "Sampling/Type" column of Table 1 aggregates several pieces of information about the strategy used in the *Sampling* task. First, we make a distinction between *selection* sampling, which is characterized by the fact that samples must belong to D ($h_\theta : E \times D \to D$) and *perturbation* sampling which can produce any element of I D ($h_\theta : E \times D \to I$). We also distinguish *deterministic* sampling, which always returns the same set of samples, and *random* sampling.

Design choices related to the *Generation* task:

- Many BEM leverage existing machine learning methods to generate an explicit proxy F_w approximating M. The "Model" column of Table 1 shows the type of "interpretable" model used by these BEMs (Trepan [4], BETA [14], Anchors [22], LIME [21], LEMNA [10], Local-Gradient [2]). Some of these methods return the interpretable model itself as an explanation while others apply a further explanation generation step, which is denoted by G_E in the "Steps" column.

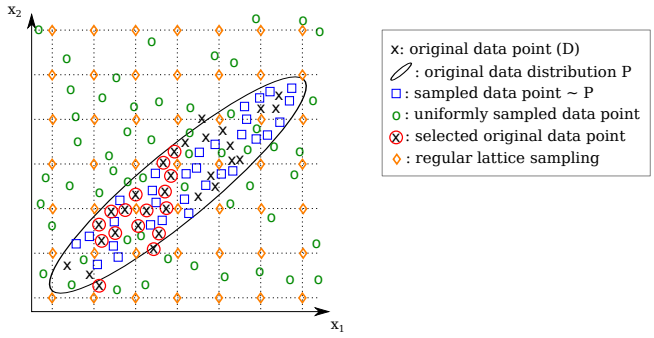


Figure 2: Illustration of sampling categories in 2 dimensions

- The "Objectives" column shows the objectives of each BEM listed by decreasing order of priority. Technically speaking, these priorities are defined through the choices of constraints, criteria and weights (respectively o_i and c_i and λ_i in Definition 4 of Section 2.3). Defining an objective as a constraint rather than a criterion is a way to assign it a higher priority. For instance, BETA [14] uses three constraints on the complexity of the rule-based proxy model and five criteria related to the fidelity of the proxy to M. Conversely, Anchors [22] sets a constraint on the fidelity of the rule-based model and criteria on the number of rules. As suggested by the titles of the papers, BETA focuses on interpretability of the explanation while Anchors puts more emphasis on fidelity.

General design choices:

- The "I/O" column in Table 1 provides information about the fact that the *Sampling* task is called iteratively (I) or is a one-shot task (O). Intuitively, the iterative mode may lead to sample sets that are more precise because they are tailored to the needs of the *Generation* task. Selecting 1000 emails in the spam filter example is an illustration of the one-shot mode. Another option could be to select only 100 samples in a first iteration step. A second iteration would focus on the region that was underrepresented in the first sample set, for example by querying only samples with long signatures. This strategy could be useful to reduce the number of model calls in the *Generation* task.

- To conclude, the "Steps" column characterizes each BEM based on the three steps identified in Section 2.2 and Section 2.3: respectively Sampling (S), model generation G_M and explanation generation (G_E). It is interesting to notice that five of the BEM in the table involve the tree steps, only one of them (Local Gradient) does not involve sampling and three of them do not have any model generation step: Shapley [29], PDP ICE [13] and QII [5] rely on specific rules to combine directly elements of S to build explanations, hence they bypass the model generation step. Also three methods do not have any explanation generation phase (they return the proxy model as an explanation).

3.2 Example: counterfactuals

Providing a counterfactual is often an effective way to help an explainee getting an intuition about the result of the ADS for a given input [27]. What we call a counterfactual here is an alternative

Name	Sampling			Generation		Output Type	Steps[b]	I/O[c]
	E	D	Type[a]	Model	Objectives			
Trepan [4]	Pop.	Pop.	P&R	Decision tree	Complexity, Fidelity	Decision tree	S G_M	I
BETA [14]	Pop.	Pop.	S&D	Rule-based model	Interpretability, Fidelity Unambiguity	Rule-based global model	S G_M	I
GoldenEye* [11]	Pop.	Pop.	P&R	Permutation	Fidelity, Interaction size	Important features interactions	S G_M G_E	I
VIN [12]	Pop.	Pop.	P&D	Permutation	ANOVA projection	Important features interactions	S G_M G_E	I
PDP ICE [13]	Pop.	Pop.	P&D	NA	NA	Dependence plot for one feature	S G_E	O
QII [5]	Indiv. or or group	Pop.	P&D	NA	NA	Var. importance	S G_E	O
Anchors [22]	$\{x_e\}$	Pop.	P&R	Rule-based model	Fidelity, Complexity, Generality	Rule-based local model	S G_M	I
LIME [21]	$\{x_e\}$	$\emptyset$	P&R	Linear model	Fidelity, Complexity	Var. importance	S G_M G_E	O
Shapley [29]	$\{x_e\}$	Pop.	P&D	NA	NA	Var. importance	S G_E	O
LEMNA [10]	$\{x_e\}$	$\emptyset$	P&R	Mixture of linear models	Fidelity, Complexity	Var. importance	S G_M G_E	O
Local Gradient [2]	$\{x_e\}$	Pop.	NA	Parzen window	Fidelity	Directions of highest slope	G_M G_E	O
Counter-factuals [27]	$\{x_e\}$	Pop.	NA	Small deviation	Target output, Distance input	Example-based	S G_M G_E	I

Table 1: Comparative table of the different black-box explanation methods. The columns correspond to the parameters of our framework with the following notation (a) S: Selection, P: Perturbation, D: Deterministic, R: Random ; (b) S: Sampling, G_M: model generation phase, G_E: explanation generation phase; (c) I: Indirect, O: One-shot. *: The sampling of GoldenEye involves mixing more than two inputs. It can be expressed in our framework through the introduction of an extra step between sampling and generation. This minor extension is described in detail in K

input x' meeting certain requirements. Typical requirements are the proximity between the input of interest x_e and the counterexample ($distance(x_e, x')$ small) and the fact that the two inputs do not lead to the same decision ($M(x_e) \neq M(x')$). In the spam filter example, a good counterfactual for an email classified as a spam could be a slightly modified version of the email (e.g. removing a single word) that would not be classified as a spam. It would establish a causal relationship between the presence of the word and the classification as spam.

Technically speaking, the computation of counterfactuals does not involve the construction of an approximation of the original model but a perturbation of input values. More precisely, we can characterize the *Generation* task by the following function:

$$F_w(x) = M(T_w(x)) = M(x + w) \tag{5}$$

where T_w denotes a translation operator such that $x + w \in I$ and w the perturbation from the original input. Here, w does not represent a parameter of a model but a parameter of a perturbation in the input space I. Interestingly, our framework is general enough to capture this case as well, as shown in the above definition. The optimization problem is the search for w^* such that:

$$\begin{aligned} w^* = \underset{w}{\text{argmin}} \quad & distance(x_e, x_e + w) \\ \text{subject to} \quad & F_w(x_e) = y' \end{aligned} \tag{6}$$

with y' an alternative output specified by the user. The generation of counterfactuals is typically

iterative. Samples can be queried one-by-one until the needs of the explainee are met. The resulting counterfactuals $x_e + w$ are returned as explanations. In our previous example, the email with a single word removed is delivered to the user.

Interestingly, different sampling strategies can be used, leading to different explanations. For example, if the explainee prefers counterfactuals that are real samples, then selection sampling would be the most appropriate. Otherwise, random sampling can be used. Using the population distribution leads to realistic samples while not using it may lead to samples that are closer to the input of interest but unrealistic. For instance, if any mail containing "Fg_f" is classified as a spam, then the *Generation* task could explain any non-spam email by adding this meaningless sequence of characters to the email. On the other hand, sampling from the population distribution could discard such unrealistic counterexamples. Therefore, the intention of the explainee and his level of expertise should be taken into account to choose the sampling strategy. A lay user may be more interested in realistic counterfactuals while learning about odd features can be valuable to the system designer for the purpose of debugging.

4 Exploration of the design space

Beyond its interest to provide a systematic overview of existing explanation methods, as described in the previous section, our framework can provide valuable help for designers of new explanation systems. As suggested in the introduction, an explanation system should ideally be interactive and allow explainees to ask different types of questions to enhance their understanding of the ADS. Depending on the type of questions, different choices can be made for the *Sampling* and *Generation* tasks.

As an illustration, we show in this section how the type of question should influence the *Sampling* task. We use a simplified version of the previous spam classifier example in which M has only one feature: the ratio r_{cap} of capital letters in an email. The output remains the probability of being a spam. The shape of M is shown Fig. 3 (a). $P_{r_{cap}}$, which denotes the probability distribution of r_{cap}, is pictured in Fig. 3 (b). We compare, as an illustration, a sampling based

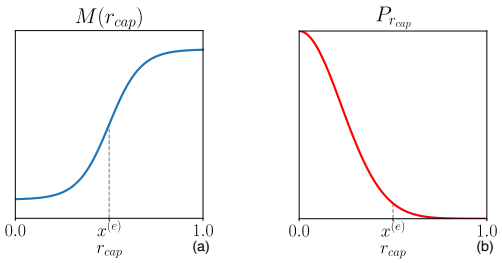


Figure 3: (a) Shape of the model $M(r_{cap})$ (b) Probability distribution of r_{cap} (c) Joint probability of $SPAM{=}1$ and r_{cap}

on $P_{r_{cap}}$ with a uniform sampling between 0 and 1. They may generate respectively the sample sets S_p and S_u:

$$\begin{aligned} S_p &= \{0.03, 0.21, 0.17, 0.1, 0.4, 0.01, 0.0\} \\ S_u &= \{0.45, 0.87, 0.15, 0.4, 0.98, 0.49, 0.54\} \end{aligned} \tag{7}$$

As expected, samples from the population distribution are mostly small values. Let us consider a local explanation: $E = \{x_e = (0.5)\}$ and the following question of the explainee: "Did the

value of r_{cap} influence the result and how (for or against the classification as a spam?)". A way to answer this question is to assess what would be the output of the model in the absence of the knowledge of r_{cap} [24]. To do so, we can compare the current value with an average situation, which suggests the use of the population distribution. Since $M(x_e) - \langle M(x)\rangle_{x\in S_p} > 0$, the probability evaluated by the model would be lower without any knowledge of r_{cap}. Therefore, this value contributed to classify it as spam. It should be noted that the result would be different if the uniform distribution were used instead because $M(x_e) - \langle M(x)\rangle_{x\in S_u} \approx 0$.

Another possible question of the explainee could be: "What would be the impact of an increase of r_{cap} on the prediction of the model?". In this case, the explanation depends on the shape of M. To be efficient and to cover the full range of values, selecting samples uniformly is a good strategy. The *Generation* task would be less efficient and the result less precise with the population distribution strategy.

5 Related work

In this section, we focus on previous proposals to define general taxonomies or classifications of explanation methods. Some of these works focus on the theoretical underpinnings of explanations while others are general overviews of existing methods. On the theoretical side, [16] introduces a formal framework to unify four BEMs. The framework is restricted to methods that compute the contribution of each feature to a given prediction. Moreover, it does not attempt to identify the common components shared by different methods.

The scope of explanation methods considered in [9] is broader, including black-box, white-box and constructive methods. This survey introduces a glossary and a taxonomy for interpretable and explainable AI. Our approach differs from [9] in several ways. First, we start from a mathematical definition of the explanation tasks[3] and we derive our classification from the parameters of the formal framework. In addition, this framework makes it possible to compare existing methods in a very precise way. The range of methods considered in [9] is broader however, as it goes beyond black-box methods. In the same vein, [1] is a high-level survey of explainable AI along four axes: explainability strategies, evaluation of explanations, interaction with humans and more general considerations about on the role of explanations.

Other surveys and taxonomies have been proposed with different focuses. For example [26] proposes a high-level taxonomy of interpretable and interactive machine learning composed of six elements[4] that are characterized in a very abstract way. Some papers focus on explanations for specific types of ML techniques. We do not discuss them in detail in this research report since our focus is black-box methods but still mention [20] which considers three types of explanation methods for deep learning[5] and discusses in a general way desirable properties of explainers and technical challenges. [23] provides a taxonomy of interpretability in Human-Agent Systems. The interest of the authors is more general as it also includes the motivations of the explainee and the expected form of interaction with the explainer. However, [23] does not compare or analyze explanation methods as it refers to a single method. [25], [15] and [28] analyze more generally the needs for explainability and transparency considering social and technical aspects. Finally, [6] provides a formal definition of explanations with a focus on the criteria to evaluate them. The evaluation of explanations, which is a critical issue, is not covered by this research report. We come back to this issue in the conclusion.

[3] Rather than the explanation *problem* as in [9], which amounts to characterize explanation tasks by their types rather than their functional definitions as done in this research report.

[4] Dataset, Optimizer, Model, Predictions, Evaluator and Goodness.

[5] They are called respectively "rule-extraction methods", "attribution methods" and "intrinsic methods".

The above papers provide very useful overviews of the field but, to our best knowledge, none of them aims to define precise technical criteria to characterize and compare on a rigorous basis existing BEM, as presented in this research report.

6 Perspectives

The main objectives of the work described in this research report are to provide a formal framework for BEMs that (i) makes it easier to compare existing approaches and (ii) can provide guidance for the design of new explanation systems. We have mostly focused on the first objective in this research report and provided some hints about the second one in Section 4.

We are currently exploring the use of this framework to build a generic explanation environment allowing BEM designers to specify their choices of parameters (as discussed in Section 2 and Section 3) in order to select appropriate components for each task.

As discussed in the introduction, explanations should be considered as interactive processes and much work remains to be done to ensure that BEM really address the needs of explainees, especially when they have little technical expertise. We believe that the systematic exploration of all possible options to build answers made possible by our framework can also be exploited to enrich the interaction with the explainee, for example by suggesting different options, or allowing him to clarify his question. Section 3.2 and Section 4 provide insights into this research direction.

Another key aspect of explanations that has not been developed in this paper is their assessment. Different criteria have been proposed to assess the quality of an explanation [6]. Our framework makes it possible to specify quality objectives, either as constraints or as criteria as presented in Section 4, but does not provide any help to evaluate the relevance of these objectives (for example through an assessment of the understanding of the explainee). This is also a major avenue for further research.

References

[1] A. Adadi and M. Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, 6:52138–52160, 2018.

[2] D. Baehrens, T. Schroeter, S. Harmeling, M. Kawanabe, and K. Hansen. How to explain individual classification decisions. page 29.

[3] J. A. Buolamwini. *Gender shades: intersectional phenotypic and demographic evaluation of face datasets and gender classifiers*. PhD thesis, Massachusetts Institute of Technology, 2017.

[4] M. Craven and J. W. Shavlik. Extracting tree-structured representations of trained networks. page 7.

[5] A. Datta, S. Sen, and Y. Zick. Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems, 2016.

[6] A. Dhurandhar, V. Iyengar, R. Luss, and K. Shanmugam. A formal framework to characterize interpretability of procedures. *arXiv:1707.03886 [cs]*, Jul 2017. arXiv: 1707.03886.

[7] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189âĂŞ1232, 2001.

[8] A. Goldstein, A. Kapelner, J. Bleich, and E. Pitkin. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *arXiv:1309.6392 [stat]*, Sep 2013. arXiv: 1309.6392.

[9] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi. A survey of methods for explaining black box models. *ACM Computing Surveys (CSUR)*, 51(5):93, 2018.

[10] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing. LEMNA: Explaining Deep Learning based Security Applications. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security - CCS '18*, pages 364–379, Toronto, Canada, 2018. ACM Press.

[11] A. Henelius, K. PuolamÃďki, H. BostrÃűm, L. Asker, and P. Papapetrou. A peek into the black box: exploring classifiers by randomization. *Data Mining and Knowledge Discovery*, 28(5âĂŞ6):1503âĂŞ1529, Sep 2014.

[12] G. Hooker. Discovering additive structure in black box functions. In *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining - KDD âĂŹ04*, page 575. ACM Press, 2004.

[13] J. Krause, A. Perer, and K. Ng. Interacting with Predictions: Visual Inspection of Black-box Machine Learning Models. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI '16*, pages 5686–5697, Santa Clara, California, USA, 2016. ACM Press.

[14] H. Lakkaraju, E. Kamar, R. Caruana, and J. Leskovec. Interpretable & Explorable Approximations of Black Box Models. *arXiv preprint arXiv:1707.01154*, 2017.

[15] Z. C. Lipton. The mythos of model interpretability. *arXiv:1606.03490 [cs, stat]*, Jun 2016. arXiv: 1606.03490.

[16] S. Lundberg and S.-I. Lee. A Unified Approach to Interpreting Model Predictions. *arXiv:1705.07874 [cs, stat]*, May 2017. arXiv: 1705.07874.

[17] P. Madumal, T. Miller, F. Vetere, and L. Sonenberg. Towards a grounded dialog model for explainable artificial intelligence. *CoRR*, abs/1806.08055, 2018.

[18] T. Miller, P. Howe, and L. Sonenberg. Explainable AI: beware of inmates running the asylum or: How I learnt to stop worrying and love the social and behavioural sciences. *CoRR*, abs/1712.00547, 2017.

[19] B. D. Mittelstadt, C. Russell, and S. Wachter. Explaining explanations in AI. *CoRR*, abs/1811.01439, 2018.

[20] G. Ras, M. van Gerven, and P. Haselager. Explanation methods in deep learning: Users, values, concerns and challenges. *CoRR*, abs/1803.07517, 2018.

[21] M. T. Ribeiro, S. Singh, and C. Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, pages 1135–1144, San Francisco, California, USA, 2016. ACM Press.

[22] M. T. Ribeiro, S. Singh, and C. Guestrin. Anchors: High-precision model-agnostic explanations. In *AAAI Conference on Artificial Intelligence*, 2018.

[23] A. Richardson and A. Rosenfeld. A survey of interpretability and explainability in human-agent systems. page 7.

[24] M. Robnik-Åăikonja and I. Kononenko. Explaining classifications for individual instances. *IEEE Transactions on Knowledge and Data Engineering*, 20(5):589–600, 2008.

[25] R. Tomsett, D. Braines, D. Harborne, A. D. Preece, and S. Chakraborty. Interpretable to whom? a role-based model for analyzing interpretable machine learning systems. *CoRR*, abs/1806.07552, 2018.

[26] E. Ventocilla, T. Helldin, M. Riveiro, J. Bae, and N. Lavesson. Towards a taxonomy for interpretable and interactive machine learning. 2018.

[27] S. Wachter, B. Mittelstadt, and C. Russell. Counterfactual explanations without opening the black box: Automated decisions and the gdpr. *SSRN Electronic Journal*, 2017.

[28] A. Weller. Challenges for transparency. *arXiv:1708.01870 [cs]*, Jul 2017. arXiv: 1708.01870.

[29] E. Åătrumbelj and I. Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and Information Systems*, 41(3):647âĂŞ665, Dec 2014.

The icon of Fig. 1 was made by turkkub from www.flaticon.com

Appendices

In the remaining, and unless specified differently, elements of the input space I are vectors in K dimensions composed of numerical and categorical values. We use the following notation:

$$x_i = (x_{i1}, ..., x_{iK}) \in I \tag{8}$$

The index i refers to the position of x_i in a list of elements of I. For example $D = \{x_1, ..., x_i, .., x_N\}$. The second subscript refers to the coordinates of the vector in the input space I: x_ij is the j^{th} coordinate of the i^{th} element of D.

A LIME

Local Interpretable Model-agnostic Explanations (LIME) was created in [21]. Basically, it fits a linear model "in the neighborhood" of the point x_e to be explained and uses the coefficients of the regression to estimate which features were the most influential for the model.

Given x_e, it creates a data set S composed of n perturbed replications of x_e by randomly setting s (random variable) features to 0. For instance with $K = 4$ and $n = 3$:

$$\begin{aligned} x_e = (x_{e1}, x_{e2}, x_{e3}, x_{e4}) &\xrightarrow{s=1} x'_1 = (0, x_{e2}, x_{e3}, x_{e4}) \\ &\xrightarrow{s=2} x'_2 = (x_{e1}, 0, x_{e3}, 0) \\ &\xrightarrow{s=2} x'_3 = (x_{e1}, x_{e2}, 0, 0) \end{aligned} \tag{9}$$

we create $S = \{x'_1, x'_2, x'_3\}$. Then, the output of each perturbed replications is computed and a weighted lasso regression is fitted on the resulting training set $\{(x', M(x')) \mid x' \in S\}$. The weights of the regression are decreasing as a function of the distance to x_e: $exp(-d(x_e, x')^2/\sigma^2)$, with σ a parameter of the model , which ensure the locality of the explanation. Finally, the coefficients of the linear model are directly used to measure the importance of feature in the prediction of x_e. Please refer to the original paper [21] for a more detailed description of the method.

Sampling:

$$E = \{x_e\} \tag{10}$$

$$D = \emptyset \tag{11}$$

Interestingly, LIME is one of the only methods that do not use a dataset apart from the point to be explained x_e. Firstly, we build the range of parameters:

$$\Theta = \left\{ \theta_j = \Big\{ \text{SRSWOR of size } s_j \text{ in } \{1, ..., K\} \Big\} \mid j \in \{1, ..., K\} \right\} \tag{12}$$

with $s_j \sim Unif\{0, K\}$, SRSWOR denotes "Simple Random Sampling Without Replacement" and $Unif\{0, K\}$ denotes the discrete uniform distribution of integers between 0 and K. θ_j is the set of features' index that are nullified. Now we can define the sampling functions[6]:

$$h_\theta(x_e, x_p) = \left(\begin{cases} 0 & \text{if } j \in \theta \\ x_{ej} & \text{otherwise} \end{cases} \right)_{j=1,...,K} \tag{13}$$

[6]It should be noted that $h_\theta(x_e, x_p)$ is a vector of I and that the index j refers to the coordinates of this vector $j \in \{1, ..., K\}$

h_θ acts like a masking function, it masks all features which index appears in θ. In (9), $\theta_1 = \{1\}$, $\theta_2 = \{2, 4\}$ and $\theta_3 = \{3, 4\}$. Then the equation (1) can be used to generate the database of perturbed examples as it is done in LIME. We can see that the sampling is a **random perturbation**.

It should be noted that in the case of images the features are not set to 0 but to the value corresponding to the color grey. In the case of text, the cardinal of x_e changes but the idea of the sampling remains the same.

Generation:

LIME uses F_w as an approximative model for M. More precisely, F_w is a linear function of the inputs:

$$F_w(x) = \sum_{i=1}^{K} w_i x_i + w_0 \tag{14}$$

Two criteria are optimized during the generation (both as criteria so we note c_1 and c_2).

$$c_1(w, S) = \sum_{x' \in S} e^{-\frac{d(x_e, x')^2}{\sigma^2}} (F_w(x') - M(x'))^2 \tag{15}$$

$$c_2(w) = \|w\| \tag{16}$$

c_1 ensures that the approximate linear model is faithful to M in the neighborhood of x_e while c_2 ensures that the resulting local model is not to complex. How should be weighted this two objectives is not clearly mentioned in the original article.

Delivering:

Once the model has been fitted, the coefficients of the linear regression are directly delivered to the user. It should be noted that in the case of image classification, it is more understandable to deliver to the user the pixels that have a positive value (pixels that contributed toward the prediction of the correct class).

B Anchors

"Anchors" [22] looks for the biggest square region of the input space that contains the initial input and which preserves the same output with high-precision. The explanation is a rule based model faithful locally around the initial input x_e. It works on an iterative basis. At each step i, the analyzer proposes a candidate set of rules A_i, for example:

$$A_0 = \text{"}3 \geq x_1 \geq 2\text{" AND "}x_3 = 0.5\text{"}, \tag{17}$$

and the rules are tested by querying many samples that verifies it. After a successful search, it can be said, for example, among elements of I that verify the rule A_0, 95% of them output the value $M(x_e)$.

Sampling:

$$E = \{x_e\} \tag{18}$$

$$D = \{x_1, ..., x_N\} \tag{19}$$

Anchors works with different types of data, which have different sampling strategies. However, the same idea remains. The parameters of the sampling θ contains the list of features that appear

in the candidate set of rule as current iteration. For example, for the anchor defined by (17), the corresponding parameters are $\theta_0 = \{1, 3\}$. Samples mix features of x_e and x_p. Features that appear in θ are taken from x_e and others from x_p. For tabular and image data, h_θ can be written in the following form:

$$h_\theta(x_e, x_p) = \left(\begin{cases} x_{ej} & \text{if } j \in \theta \\ x_{pj} & \text{otherwise} \end{cases} \right)_{j=1\ldots K} \tag{20}$$

For text classification, the sampling is more elaborate. If $x_i \in I$ is a sequence of words, then x_{ij} denotes the j^{th} word in the sentence. θ still contains the list of words that appear in the candidate set of rules:

$$h_\theta(x_e, x_p) = \left(\begin{cases} x_{ej} & \text{if } x_{ej} \in \theta \\ x_{pk} & \text{with proba } p \text{ otherwise} \\ x_{ej} & \text{with proba } (1-p) \text{ otherwise} \end{cases} \right)_{j=1\ldots|x_e|} \tag{21}$$

with x_{pk} such that x_{pk} and x_{ej} have the same POS tag. And with p proportional to their similarity in an embedding space. In both cases, the parameter space only contains one element $\Theta = \{\theta\}$. This sampling does not replace systematically every word absent from the rule but with a probability. Moreover, instead of replacing the i^th word of x_e with the i^th word of x_p, which would be meaningless in most cases, a word with the same function in the sentence is used (it replaces a verb with a verb or a noun with a noun). This sampling is an efficient way to generate, from the distribution of the data, many realistic samples that verify the candidate rule.

The optimization method employed by Anchors to minimize the number of model call requires that the samples are generated one-by-one (for the case of precision estimation, see below). The sample is generated with the parameter θ, the input to be explained x_e and one sample from D selected randomly. The random selection of a sample is made thanks to the filter function F.

$$F(\theta, x_e, x_p) = \mathbb{1}_{\text{index}(x_p)=r} \tag{22}$$

with $r = Unif\{1, N\}$ (drawn at each new sampling). The stochastic nature of the sampling of Anchors comes from this drawing. The sampling of Anchors is hence **random** and **perturbative**.

Generation:

Anchors is a typical generation case. F_w is an interpretable model used to approximate M. More precisely, F_w is a rule-based model (RBM) and the rules are represented by w. We use the notation $w(x) =$ True when x verifies the rule defined by w (false otherwise).

$$F_w = RBM_w \tag{23}$$

The goal of the optimization is to find the set of rules that cover most samples while satisfying probabilistic fidelity constraints. To evaluate the coverage, an extra sampling step is required with $\theta = \emptyset$ ($\Theta = \{\emptyset\}$), we call the result of the sampling $S_\emptyset$. It should be noted that, except for the case of text data, $S_\emptyset = D$. The following criterion:

$$c(w, D) = -\frac{\sum_{x \in S_\emptyset} \mathbb{1}_{w(x)}}{|S_\emptyset|}, \tag{24}$$

ensures that the rule coverage is as big as possible (ratio of inputs verifying the rule among the data set D). And the constraint on the fidelity can be expressed as:

$$o(w, D) = \mathbb{P}\Big(\frac{\sum_{x' \in S} \mathbb{1}_{F_w(x')=M(x')}}{|S|} \geq \tau \Big) \geq (1 - \delta) \tag{25}$$

with τ and δ parameters of the method. This constraint, ensure that the rule is verified at a certain precision (τ) with a chosen level of uncertainty (δ). In the theoretical formulation, the size of S is arbitrary. The optimization problem can be formulated as in (4). The optimization process is **iterative**. Each step of Anchors consists of 2 iterations in our framework: one iteration for each coverage estimation and one iteration for each precision estimation.

Delivering:
Anchors does not require a delivering step as the explanation is the model F_w itself.

C Shapley values

In [29], the authors are using Shapley values, a result from cooperative game theory, to build a measure of the contribution of a feature for a specific classification (local explanation). Shapley's result is interesting because it has four mathematical properties which ensure consistency with what is expected of a feature contribution. We note x_e the point to be explained, as in [29] we can define for any $Q \subseteq \{1, ..., K\}$

$$M_Q(x_e) = \mathbb{E}\Big[M(X_1, ..., X_K) \mid X_i = (x_e)_i, \forall i \in Q\Big] \tag{26}$$

$$\Delta_Q(x_e) = M_Q(x_e) - M_{\{\}}(x_e) \tag{27}$$

Then, the importance of the variable i for the data point x_e is given by the Shapley value:

$$\phi_i(x_e) = \sum_{Q \subseteq [1,...,K] \setminus \{i\}} \frac{|Q|!(K - |Q| - 1)!}{K!} \Big(\Delta_{Q \cup \{i\}}(x_e) - \Delta_Q(x_e)\Big) \tag{28}$$

Please refer to the original paper for more details on the meaning of this formula.

Sampling:

$$E = \{x_e\} \tag{29}$$

$$D = \{x_1, ..., x_N\} \tag{30}$$

The sampling mixes features of x_e with features of x_p to generate a new samples. The parameter of the sampling defines which features from x_e are used ($i \in Q$, in equation (26) and which feature from x_p are used ($i \notin Q$, in equation (26). Hence, the parameters of the sampling θ contain the list of features that appear in the tested combination Q: $\theta = Q$. Samples mix features of x_e and x_e. Features that appear in θ are taken from x_e and others from x_p.

$$h_\theta(x_e, x_p) = \left(\begin{cases} x_{ej} & \text{if } j \in \theta \\ x_{pj} & \text{otherwise} \end{cases} \right)_{j=1...K} \tag{31}$$

Then, we define the range of parameters:

$$\Theta = \Big\{Q \mid Q \subseteq \{1, ..., K\}\Big\} \tag{32}$$

We can see that the sampling if a **deterministic perturbation** of the inputs.

Generation:

There is no generation step for this method.

Delivering:
We now assume that (26) is best approximated by the empirical mean:

$$M_Q(x_e) \approx \frac{1}{|S|} \sum_{x' \in S} M(x') \tag{33}$$

with S defined by (1). We can now see that the Shapley value can be approximated by averaging the dataset defined by (1), with the weights of (28). The approach is **direct**.

D Quantitative Input Influence

Quantitative Input Influence (QII) [5] describes how to compute the degree of influence of inputs on outputs of black-box in several cases. In particular, it can explain outputs about individuals and about group (by measuring the disparate impact of a features among groups). Moreover, it allows the computation of joint influence (how several features influence output) and of marginal influence. Using Shapley's formula as an aggregation of QII is also presented.

Technically speaking, the QII are based on the equation (26). The differences between explanations are obtained during the delivering step.

Sampling: The sampling of QII is strictly equivalent to the sampling of Shapley presented in the section C.

Generation:
There is no generation step for this method.

Delivering:
We show with some examples from [5] of how the result of (26) can be use to explain a black-box. First, as in [24], (26) can be used to estimate the output of the black-box in this absence of the knowledge of a variable for a specific individual x_e. This can be estimated with for the feature i with:

$$M_{\{1..K\}}(x_e) - M_{\{1..K\}\setminus\{i\}}(x_e) \tag{34}$$

This formulation is equivalent to the equation (3) of [5]. To compute the influence on groups, the latter is averaged over all elements of the groups. If we name Y the group then the impact of i on Y is given by:

$$\mathbb{E}_{x \in Y}\Big[M_{\{1..K\}}(x_e) - M_{\{1..K\}\setminus\{i\}}(x_e)\Big] \tag{35}$$

The interested reader may find other options in the original publication.

E PDP and ICE

Partial Dependence Plot (PDP) was introduced in [7] and Individual Conditional Expectation (ICE) in [8] and both were revised in the lens of explanation in PDP ICE [13], which is the version presented here.

It is a feature-wise global explanation method to estimate the global impact of a feature i over its range of values. The output is a plot whose x-axis is all possible values of i and y-axis the

average value of the model M over S with the value of the i^{th} fixed (PDP) or the superposition of every sample of S with modified value for i (ICE).

In both cases, the sampling is a **deterministic perturbation** over the full range of values. This method is a global explanation method in which $E = D$.

Sampling:

$$E = \{x_1, ..., x_N\} \tag{36}$$

$$D = \{x_1, ..., x_N\} \tag{37}$$

In this case, the roles of E and D are identical, which means that one of the data set is not needed. In the following, we artificially set $D = \{0\}$. It should be noted that it could have been done with our framework by using F to filter unneeded elements.

PDP ICE is feature specific, we name i the feature that is explained. If i is a continuous value, we can divide its range in r regular steps.

$$l_i = \min_{x_k \in D} x_{ki}, \quad u_i = \max_{x_k \in I} x_{ki} \tag{38}$$

$$h_\theta(x_e, 0) = \left(\begin{cases} l_i + \theta \dfrac{u_i - l_i}{r-1} & \text{if } j = i \\ x_{ej} & \text{otherwise} \end{cases} \right)_{j=1...K} \tag{39}$$

$$\Theta = \{0, 1, ..., r-1\} \tag{40}$$

If the feature i is categorical, we note $\{v_1, ..., v_r\}$ the possible values:

$$h_\theta(x_e, 0) = \left(\begin{cases} \theta & \text{if } j = i \\ x_{ej} & \text{otherwise} \end{cases} \right)_{j=1...K} \tag{41}$$

$$\Theta = \{v_1, ..., v_r\} \tag{42}$$

We can now generate the sample set S using (1).

Generation:

There is no generation step for this method.

Delivering:

The explanation can be directly computed from the set $\{(x, M(x))x \in S\}$ by computing the proper average or by displaying all curves on the same plot.

F BETA

Black box Explanation Through transparent Approximation (BETA) [14] is a global explanation method that approximates the model by an interpretable rule-based model. Selected model tries to minimize 8 criteria through an optimization procedure. The optimization is **iterative** and a candidate rule-set is evaluated at each iteration.

Sampling:

$$E = \{x_1, ..., x_N\} \tag{43}$$

$$D = \{x_1, ..., x_N\} \tag{44}$$

Similarly to E, we artificially set $D = \{0\}$

The sampling parameters θ define a candidate set of rules. For ease of notation, we use $\forall x \in I$, $\theta(x)$ is Boolean value (true if x verifies the rule defined by θ false otherwise). Then the sampling is defined by:

$$h_\theta(x_e, x_p) = x_e \tag{45}$$

$$F(\theta, x_e, x_p) = \mathbb{1}_{\theta(x_e)} \tag{46}$$

The sampling of BETA is **selective** and **deterministic**, it simply selects all examples from D that satisfy the rule defined by θ.

Generation:

The generation function is an approximative model which tries to maximize the fidelity with M and other criteria. The approximative model is a rule based model: $F_w = RBM_w$ and the criteria to optimized are explicitly mentioned in the article (cf section 2.2 of [14]). One can simply verify that every criteria appearing in the optimization problem of BETA (cf. equations (1) and (2) of [14]) can be expressed analytically w.r.t w and S.

Delivering:

There is no need for a delivering step as the rule-based model itself is provided as an explanation.

G LEMNA

"Local Explanation Method using Nonlinear Approximation" (LEMNA) [10] is a local explanation method. It is based on the framework as LIME but uses a mixture regression model with a fused Lasso regularization.

Sampling:

The sampling part of LEMNA is similar to LIME as it is described in A. [10] does not give a lot of details on the sampling procedure used by their method. The sentence "The idea is to randomly nullify a subset of features of x" suggest that the sampling function used is defined as in (13). But the absence of weights decreasing with the distance to x_e and the sentence "we first synthesize a set of data *samples locally (around x)*" suggest that the choice of sampling parameters enforce locality.

Generation:

The generation step is based on an approximation of M in the neighborhood of the data point to be explained x_e. The generation function is a Mixture of Linear Models (MLM) $F_w = MLM_w$. Similarly to LIME, the criteria to optimize are the fidelity of the approximation model and a regularization:

$$c_1(w, S) = \sum_{x' \in S} (F_w(x') - M(x'))^2 \tag{47}$$

$$c_2(w) = \|w\|_{fussed} \tag{48}$$

c_1 ensure that F_w is faithful with M, while c_2 ensure that the model is not too complex. The fidelity criterion c_1 is classic but the complexity criterion c_2 is tailored for application with data having a sequential structure (please refer to the equation (7) of the original article for more

details).

Delivering:
As for LIME A, the coefficients of the linear regression model are used as explanation. Unfortunately, few information is provided on which coefficients, among all linear models, should be chosen.

H VIN

Variable Interaction Network (VIN) [12] is a global explanation method based on the theory of Analysis of Variance (ANOVA) which aims at detecting additive structure in a black-box. Two features are additive if they do not interact for the prediction. As Shapley values, VIN combines partial permutations of the samples to detect the influence of specific features or groups of features. But thanks to a clever formulation of the problem, VIN does not require a number of model estimations exponential in the number of features which make it practical for more applications.

The VIN algorithm uses an iterative approach. While evaluating the importance of some combinations of features, VIN gather formal guaranties of smallness for other combinations and can thus avoid the computation of these. The computation of important features interaction involves a complex analytic formula. It is out of the scope of this research report to show or explain this formula in details. We simply show that each term of the formula can be extracted thanks to the sampling procedure. Then, the result can be computed with S. The quantity of interest is a sum of terms of the form:

$$\mathbb{E}_{-v}[M(x)] = \frac{1}{N}\sum_{i=1}^{N} M(x_v, x_{i,-v}) \tag{49}$$

where $v \in \{1, ..., K\}$, $x_v = (x_i | i \in v)$ and $x_v = (x_i | i \notin v)$.

Sampling:

$$E = \{x_1, ..., x_N\} \tag{50}$$

$$D = \{x_1, ..., x_N\} \tag{51}$$

As for Shapley C and QII D, sampling mixes the features of the elements x_e and x_p. The parameters of the sampling θ contains the list of features that appear in the tested combination v: $\theta = Q$. Features that appear in θ are taken from x_e and others from x_p.

$$h_\theta(x_e, x_p) = \left(\begin{cases} x_{ej} & \text{if } j \in \theta \\ x_{pj} & \text{otherwise} \end{cases} \right)_{j=1...K} \tag{52}$$

Generation:
The generation function of the VIN algorithm is a perturbation of M by permutation of features of input as mentioned in (49). The rest of the generation only involves summation and iterative search over a lattice of subset. We invite the interested read to refer to [12] for more details. We do not give the details of the computation of the projection of M, instead we show that the basic component can be obtain with the following generation function:

$$F_w(x) = \frac{1}{|S'|}\sum_{x \in S'} M(x) \tag{53}$$

with $S'(x,w) = \{h_w(x,x_p) \mid x_p \in D\}$ (derived from 1).

I Local-gradient

The method proposed in [2], which we named "Local-Gradient", in a local explanation method of complex classifier. The basic assumption of this method is that a classification can be explained by pointing the direction of highest slope toward another class. With this assumption, a classification can be explained by estimating the gradient at a given point.

Technically, this method approximate M with a kernel density approximation method (Parzen windows) based on the data set available to the explainer D. Parzen windows are then used to compute gradients in the neighborhood of the point to be explained.

Sampling:

$$E = \{x_e\} \tag{54}$$

$$D = \{x_1, ..., x_N\} \tag{55}$$

Parzen windows are directly learned on D. Hence, there is no sampling for this method. It should be noted that the authors mention the possibility of sampling new data points if needed.

Generation:

In [2], model selection and model fitting are described extensively. We limit ourselves here to the main components. The labels of the classifier are defined by: $\{1, ..., C\}$. We define the index set: $I_c = \{i \mid M(x_i) = c\}$. The main building bloc of F_w is the following:

$$\hat{p}_\sigma(x,c) = \frac{1}{n} \sum_{i \in I_c} k_\sigma(x - x_i) \tag{56}$$

with $\sigma \in \mathbb{R}$ and $k_\sigma(x) = exp(-0.5x^T x/\sigma^2)/\sqrt{2\pi\sigma^2}$. Which is used to estimate the conditional probability:

$$\hat{p}_\sigma(c|x) = \frac{\hat{p}_\sigma(x,c)}{\hat{p}_\sigma(x,c) + \hat{p}_\sigma(x, y \neq c)} \tag{57}$$

Finally, the function used to approximate the classifier M is given by:

$$F_\sigma = \underset{c \in \{1,...,C\}}{\arg\min}\ \hat{p}_\sigma(x, y \neq c) \tag{58}$$

Then, the optimization problem is given by:

$$c(\sigma, S) = \sum_{i=1}^{N} \mathbb{1}_{M(x_i) \neq F_\sigma(x_i)} \tag{59}$$

which is a simple minimization of a fidelity criteria. It is expected that F_σ is as close as possible to M on the points of D. It should be noted that although it is a local explanation method, the approximate model is fitted on the whole data set D. Thanks to this approach, the same F_σ can be used to explain any individual classification without generating a new model. On the other hand, because M can be arbitrarily complex, F_σ may not be very accurate everywhere.

Delivering:

The last step of the explanation process is the computation of the derivative of F_σ at the point of explanation. F_σ was chosen explicitly such that the derivative are easily computable. The derivative can be computed analytically with the formula of the Definition 3 of [2]. The derivatives are used as local explanations.

J Trepan

Trepan [4] is a method to approximate a black-box classifier with a decision tree. Because it is composed of rules that are human-readable, this approximate model offers an interpretable simplified version of the classifier. Broadly speaking, Trepan is a global explanation method that uses an approximative interpretable model to approximate M over all inputs of D. But, the strength of Trepan relies of the use of sampling to focus on imprecise parts of the model. Trepan uses the classical top-down approach to build the decision tree. The node at which there is the greatest potential to increase the fidelity is chosen to be the next splitting node. If the latter does not contain enough samples (S_{min}, the minimum number of samples, is a parameter of the method) then extra inputs are sampled from I such that they "fall" in this specific node.

Sampling:

$$E = \{x_1, ..., x_N\} \tag{60}$$

$$D = \{x_1, ..., x_N\} \tag{61}$$

In a nutshell, the sampling of Trepan is **random** and **perturbative** and draws a fixed number of samples that satisfy some rules of the decision tree. The sampling of Trepan differs from other methods because the distribution of the population is not directly estimated through sampling, instead it is done indirectly by using D to estimate a distribution function and then drawing from this distribution function. The modeling of each feature's distribution is done before the sampling. For discrete-values features, frequency counts are used while kernel density estimation methods are used for continuous features. For each feature i, we note P_i the estimated distribution.

Trepan needs to select inputs that respect certain rules of the decision tree. The sampling parameter θ is used to account these rules. We say that $\theta(x) = 1$ if x verifies the rule defined by θ. Then the following sampling function can be used:

$$h_0(x_e, x_p) = (x_j \sim P_j)_{j=1...K} \tag{62}$$

along with the filter:

$$F(\theta, x_e, x_p) = \theta(h_0(x_e, x_p)) \tag{63}$$

It should be noted that, unlike most of other methods, features are drawn independently.

Generation:

The generation of Trepan involves an decision tree which is fitted to minimize the distance with the outputs of M. The generation function is noted $F_w = DT_w$ with w the parameters of the decision tree. A constraint is added on the size of the tree to ensure that the resulting explanation is understandable. The optimization problem can written in the following way:

$$c_1(w, S) = \sum_{x \in S} fidelity(M(x), F_w(x)) \tag{64}$$

$$o_1(w) = size(DT_w) \tag{65}$$

The complexity of the explanation is controlled, through a hard constraint o_1, with the size of the decision tree (total number of nodes in the tree).

Delivering:
The model F_w itself is given as the explanation. Thus there is no explanation generation for this method.

K GoldenEye

GoldenEye [11] assumes that evaluating the impact of a single variable is not enough to understand the complete behavior of a classifier. Therefore, it focuses on selecting the groups of features that are globally the most important for the classification.

GoldenEye evaluates the global importance of a feature by observing how the model is modified when the values of the features are randomly permuted among the dataset. Interaction of features are evaluated by permuting simultaneously groups of features. Finally, GoldenEye defines an original way of permuting samples among the dataset:

- within class: permutation of values between two samples is permitted only if the two samples have the same model's output,
- fully random: permutation of values is allowed between any two samples.

GoldenEye looks for the grouping of features (a set of disjoint groups of features) which permutation least modifies the output of the model for the whole data set.

The sampling of GoldenEye is rather complex. We should first define a grouping of features: $G = \{S_1, ..., S_m\}$ s.t. $S_i \subseteq \{1, ..., K\}$ and $\forall i, j,\ S_i \cap S_j = \emptyset$ (cf. definition 4 of [11]). Each S_i is a group features (ex: $S_0 = \{1, 3\}$ denotes the features indexed by 1 and 3). Features are gathered in groups to test if they interact. Basically, as for Shapley, QII or Anchors, samples are generated by combining features from different elements of D. To test interactions, two features that appear in the same group are permuted together; the values of the features 1 and 3 after the permutation come from the same element. Moreover, features that appear in no group are also permuted but alone. Finally, features that belong to a group are permuted with-in class; the values of the features 1 and 3 after the permutation come from the same element and this element has the same output than the initial one. Let's take an example from a sample x_e in five dimensions:

$$x_e = (x_{e1}, x_{e2}, x_{e3}, x_{e4}, x_{e5}) \tag{66}$$

with the grouping of features: $\Big\{\{5\}, \{1, 3\}\Big\}$. To create a sample from x_e, 4 other samples are needed: $x_{j_1}, x_{j_2}, x_{j_3}, x_{j_4}$, with:

$$j_1, j_2, j_3, j_4 \sim Unif\{1, N\} \tag{67}$$

N being the size of D. The first two samples x_{j_1} and x_{j_2} are used for permuting the features of the group while x_{j_3} and x_{j_4} are used for permuting the features that don't appear in the grouping (here 2 and 4). The following constraints are applied to j_1 and j_2.

$$M(x_{j_1}) = M(x_{j_2}) = M(x_e) \tag{68}$$

This constraint specifies that two samples (for groups appearing in the grouping) must be drawn within class. After these operations, the final sample is:

$$(x_{j_2 1}, x_{j_3 2}, x_{j_2 3}, x_{j_4 4}, x_{j_1 5}) \tag{69}$$

We see that 4 samples are needed to create a single sample from x_e. Because our framework only allows the combination of two samples x_e and x_p, it is not straightforward to describe the sampling of GoldenEye with our framework. A solution is to draw distinct samples with our framework and then to combine them during an additional step that occurs between the sampling and the generation. In our example, we draw 4 samples:

$$\begin{aligned} j_1 &\to (0, 0, 0, 0, x_{j_1 5}) \\ j_2 &\to (x_{j_2 1}, 0, x_{j_2 3}, 0, 0) \\ j_3 &\to (0, x_{j_3 2}, 0, 0, 0) \\ j_4 &\to (0, 0, 0, x_{j_4 4}, 0) \end{aligned} \tag{70}$$

and we then recombine them to obtain the sample of (69). It should be noted that the description of GoldenEye forces us to bend the rules of the framework a little. To our knowledge, GoldenEye is the only method to use such kind of sampling. If the interest of this type of sampling is proven in practice, we should consider adapting our framework to better take into consideration this strategy.

Sampling:

The sampling of GoldenEye is a **random perturbation** of samples. The random permutation with the condition is drawn in advance and the indices are stored in Θ. Each element of Θ is composed of 4 values:

1. index of x_e in the E,
2. type of permutation (fully random "fr" or within class "wi"),
3. the features concerned (e.g. $\{1, 3\}$),
4. the random index (other sample whose values are taken from for the permutation).

To generate the first sub-sample (j_1) of our previous example, the following set of parameters is used:

$$\theta = (1, \text{“wi”}, \{5\}, j_1) \tag{71}$$

where 1 is the index of x_e that was chosen as an example. The second parameter is "wi" because the permutation is done within class as the feature 5 appears in the grouping. $\{5\}$ is the set of features that are replaced in this sample. Finally, j_1 is the random index as previously. In the remaining, we refer to the coordinates of θ with brackets: $\theta(1) = 1$, $\theta(2) =$ "wi", ... We can now define the sampling function h_θ.

$$h_\theta(x_e, x_p) = \left(\begin{cases} x_{pj} & \text{if } j \in \theta(3) \\ 0 & \text{otherwise} \end{cases} \right)_{j=1,\dots,K} \tag{72}$$

along with the filter:

$$F(\theta, x_e, x_p) = \mathbb{1}_{index(x_e)=\theta(1)} \mathbb{1}_{index(x_p)=\theta(4)} \tag{73}$$

this filter outputs exactly one sample per set of sampling parameter θ.

Now let's build the set of parameters Θ. We start from the grouping of group $\{S_1, ..., S_m\}$ and we add $\{R_1, ..., R_p\}$ the features that do not appear in any S_i. We have $\cup_i S_i \bigcup \cup_j R_j = \{1, ..., K\}$. Then, for each sample x_e of E, we should draw $m + p$ samples from D. m of them must satisfies the within class constraint while p of them don't. So we draw randomly:

$$\forall (i, j) \in \{1..N\} \times \{1..m\},\ s_{ij} \sim Unif\{1..N\} \ \ s.t \ \ \forall (i, j),\ M(x_i) = M(x_{s_{ij}}) \tag{74}$$

and:

$$\forall (i,j) \in \{1..N\} \times \{1..p\},\ r_{ij} \sim Unif\{1..N\} \tag{75}$$

We can create the set sampling parameters for the generating the first sample:

$$\{(1, \text{“wi”}, S_1, s_{11}), ..., (1, \text{“wi”}, S_m, s_{1m}), (1, \text{“fr”}, R_1, r_{11}), ..., (1, \text{“fr”}, R_p, r_{1p})\} \tag{76}$$

and generalize it to create the full permutation of the data set:

$$\Theta(G) = \{(i, \text{“wi”}, S_1, s_{i1}), ..., (i, \text{“wi”}, S_m, s_{im}), (i, \text{“fr”}, R_1, r_{i1}), ..., (i, \text{“fr”}, R_p, r_{ip}) \mid i = 1..N\} \tag{77}$$

Finally, to obtain the desired samples elements of S should be grouped by value of $\theta(0)$ and summed as shown in the example. At the end of this operation we have generated one sample per element of E, hence we use the notation:

$$\overline{S}(G) = \{(x_e, \overline{h}_\Theta(G)(x_e)\} \tag{78}$$

where $\overline{h}_\theta(x_e)$ is the result of the composition described in the previous paragraph.

The instantiation of the sampling of GoldenEye in our framework is somehow laborious. It should first be noted that The strategy employed is complex, it is hence normal to have a complex description. Moreover, as already mentioned, our framework has to be bend in order to account for this exception.

Generation:

The generation step of GoldenEye uses a perturbative approach to detect the combinations of features that most impact globally the output of the model. The method iteratively selects the biggest combination that impact the least the output. To do so, we compare the output before the permutation with the value after the permutation. Hence the generation function is simply:

$$F_G(x) = M \circ \overline{h}_\Theta(G)(x) \tag{79}$$

and the criteria:

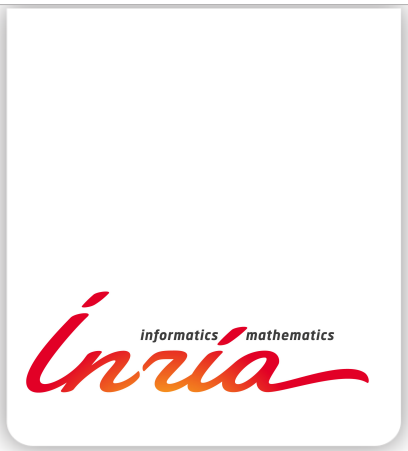
$$o_1(G, E) = \sum_{x_e \in E} \mathbb{1}_{M(x_e) = F_G(x_e)} > threshold \tag{80}$$

$$c_1(G) = -\max_{g \in G} |g| \tag{81}$$

The first equation is a hard constraint that fixes a threshold for the fidelity. Bellow this threshold the group cannot be considered as an important group. The second is an a soft criteria thanks to which the method focus on combinations of many features.

Delivering:

The delivering step of GoldenEye consist in gathering the groups of features that were selected iterativelly and to display them to the user.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399