UNIVERSITÀ DI PISA

Department of Computer Science and Informatics

Master degree in Business Informatics

Xlib - Explanation Library for Black Box Models

Supervisors:

Fosca Giannotti

Riccardo Guidotti

Anna Monreale

Candidate:

Riccardo Affolter

April 27, 2019

Contents

List of Tables

Abstract

Many machine learning algorithms are becoming a useful computational tool to find answers to support decisions, however they are strongly criticized for the lack of interpretability in relation to the predictions they provide. In favour of the compromise between accuracy and interpretability, many proposals have been found with many different approaches already developed. This thesis focuses on the creation of a stable and usable platform that works exploiting explanation methods existing in literature. In order to do this, it is necessary to explore proposed methods dealing with this topic and insert them into a platform. This platform is called Xlib which is a Python package necessary for our purpose. Our approach could be applied to give transparency in the decision policies of some predictive models that might be used by experts and non-experts in the field of machine learning. To support developers, researchers and practitioners, Xlib have to allow easy and standardized access to a wide variety of algorithms integrated into a common framework, to evaluate, compare and visualize the results they provide.

Chapter 1

Introduction

Machine learning has become an increasingly easy important area of research for its capability to retrieve decisions with an high degree of accuracy. For this reason, we are encountering a growing number of companies which are intended to apply complex data mining algorithms to reach its goal in different fields. These algorithms are called "black boxes", they are machine learning algorithms from whom it is difficult to understand the internal structure mechanisms that produce a certain result. In many cases, an application that exploits these algorithms makes a biased decision. For example, in fields like criminal justice, the results of the predictive algorithms used to get the recidivism has been perceived as distorted, due to criminogenic factors. The analysis of '*Criminal Justice and Behavior*' journal [5] reveals that the error of the prediction may be imputed to the discrimination between white and black defendants. Another example happened some years ago when Amazon.com built a software to calculate which areas of the U.S could be selected for the "Free same-day delivery" program[1]. There's no evidence that Amazon makes decisions on where to deliver based on race, but unintentionally the algorithm leads to a choice, restricted only to a certain neighborhood participating in the program [2]. Such an inequality could mean undisclosed bias or arbitrariness in the weight of the processed data. Some studies would have considered a high number of domains where the external decision made by Machine Learning-systems are affected by potential biased [6, 10, 12]. One reason for this problem is due to human bias and prejudices that significantly affects the decision-making. It is important to note that human bias exists and is hard to mitigate. Thus, bias awareness would be needed, when we are evaluating the

[1] https://www.retailmenot.com/blog/same-day-delivery.html

[2] http://www.techinsider.io/how-algorithms-can-be-racist-2016-4

behaviour of a black box. Human bias is not the only reason for this type of error. Some trivial mistake are due to the shape of the examined data. For example in the experiments conducted in a military context [13], the classifier is able to predict friendly tanks from enemy tanks, discriminating them through the colours of the sky (sunny vs. overcast days). Understanding why this type of decision is taken is really important in sectors such as the medical industry where making a bad diagnosis might lead to high reimbursement and liability [18].

There are already presents tools [16] able to provide type of explanation, in order to understand which are the main features that influence the final choice of a decision system based on ML. The grade of comprehensibility of a given formulation of explanation can be quantified in term of interpretability. Humans typically prefer an explanation that is simple to understand and to explain (deductively) [20]. We are able to find the ease of the explanation by relying on the length of the explanation considering that the longer it is, the harder it is to memorize. However, on the other hand, we also have to consider that humans have different ways of thinking, and longer explanations might be more meaningful and rational. In the last three years there has been a rapid rise in the proposal of XAI (Explainable Artificial Intelligence) resources [8] but many studies were carried out already in the early 2000s [37]. The common goal of these proposals is to provide methods, tools and mechanisms to make the decision choice system understandable. Some tools and methods [3] allow for the analysis of the correlation among features and the target to understand their contribution to the final decision. Other methods provide explanations for the decision of single instances, while others are able to explain the global behavior [7] of a black box. Some explanation methods are suitable for a specific black box [32], others are model-agnostic [26, 27, 2, 14]. The most limitation of the available tools and methods lies in the lack to provide a standard output. Each algorithm return to the users not only with different explanation formulation but also with different evaluation metrics. Moreover, any approach uses different information to explain the decision. What is needed, and also missing, is a framework that works with all these explanation methods, having tools for explanation visualization and evalutation metrics.

In this thesis we propose Xlib (eXplanation library) which is a framework that includes methods for the explanation of black box models for images and tabular data. The training phase of the dataset is geared towards different classification techniques. In this manner is possible to explore the data carefully and build various models. After the preparation of the models, it is possible to analyze the case study and all the evaluations concerning the black boxes. Then,

according to the prescribed goal, some Python extensible functionalities have been designed in order to get rid of the learned structures of a black box model both globally (inference analysis on the basis of the data) and locally (exploring an individual prediction), exploiting different approaches and being able to use new ones. Xlib provides a powerful tool that shows a clear advantage over other XAI tools because it is based on a standardized data model built for an extensible number of black boxes and explainers. In the current version of the XAI library, we have integrated four types of classifiers (Support Vector Machine, Neural Network, Random Forest, Convulation Neural Network), three local explainers (LIME,LORE,Anchor) for tabular data, three for image data (LIME,DeepExplain,Saliency) and one global explainer for tabular data (Skater). The structure of this work is organized beginning with an overview of the concept of black box, significance of interpretable methods and already available XAI tools. Next, in Chapter 3 we explain in the details the theory and the logic that covers all the explainers implemented in the framework. In Chapter 4, we then present the general structure of our work, discovering the python platform provided, listing all the algorithm scripts that guarantee extensibility. Extensibility is an important concept of our work, especially since it ensures the possibility to be usable/extensible by other developers. In Chapter 5, we validate our approach showing a possible use of the framework,reporting and comparing the results of our experiments. Finally, Chapter 6 presents the conclusions of this thesis, indicating future works.

Chapter 2

Related Work

This chapter is dedicated to a general overview of the state-of-the-art methods that explain the predictions of a black box.

People with different backgrounds are approaching differently to an increasing interest field as Machine Learning. The goal of many data scientists is to find an answer for decision support, choosing the optimal ML algorithm depending on features such as the amount of data required to train, speed, forecast accuracy or how easy is it to implement. Adding complexity to these algorithms, though, it is much harder to understand the reason that leads to a correct or wrong prediction. This leads to a lack of transparency of the decision taken and causes the necessity to discover certain techniques that enhance interpretability. For this reason, it is necessary to separate the significance of accuracy, which concerns the ability of a model to make correct predictions, with interpretability that concerns how much the model allow for human understanding. The type of explanation is divided in different categories, one regarding how the decision system returns certain results called 'black box explanation' and the second one, to find a transparent design of the classifier that made the decision on the same classification problem. Moreover, the category of black box explanation is divided into three categories: one regards the '*model explanation*' where the whole logic of the classifier is globally explained. The second one is '*outcome explanation*' and finds which are the main reasons of the prediction of a single instance with its result(or target). The last type of explanation is the '*model inspection*' where the goal, this time, is to understand the behavior of the black box when the values of the instance change [28]. In this thesis, we will focus on "black box explanation".

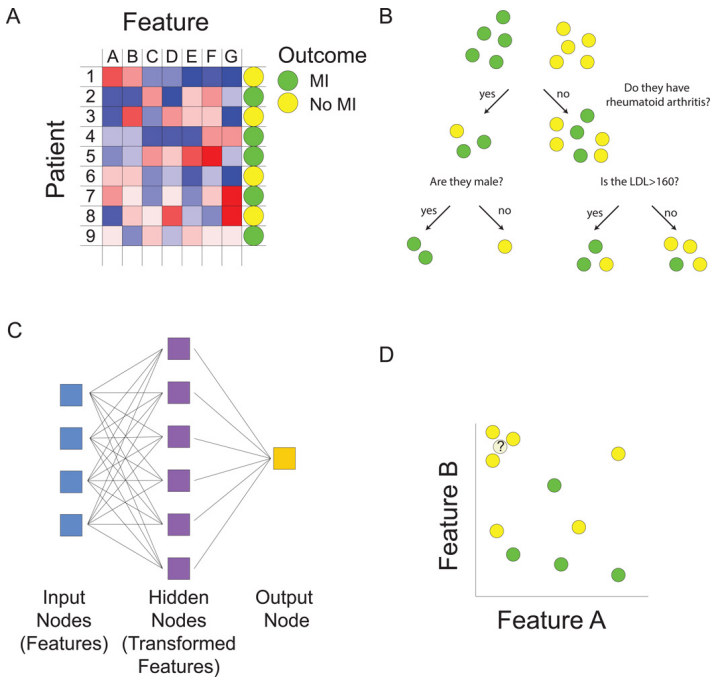


Figure 2.1: An overview of a machine learning algorithm. **A.** Represents a matrix of the supervised and unsupervised learning problem. **B.** Decision trees map features to outcome. **C.** A hidden layer of nodes integrates the value of multiple input nodes to derive transformed features. (Neural Network) **D.** Multiple types of functions can be used for mapping features to outcome. SVM or KNN [9]

Nowadays, people know that finding the hidden answer behind a data survey is increasingly important. Some machine learning algorithms require more dimensions to obtain a good accuracy but they lack in interpretability. We can gain in understandability by decreasing the accuracy of the model, despite some models are too complex and *'opaque'*. There are many comparable methods in which we can investigate on the trade-off between accuracy and interpretability.

2.1 Explaining Black Boxes

Black boxes are models that exploit algorithms, even sophisticated, to produce a prediction given a specific input. The machine learning model predicts the target value y_i using an instance $x^{(i)}$.

$$\hat{f}\left(x^{(i)}\right) = y_i \tag{2.1}$$

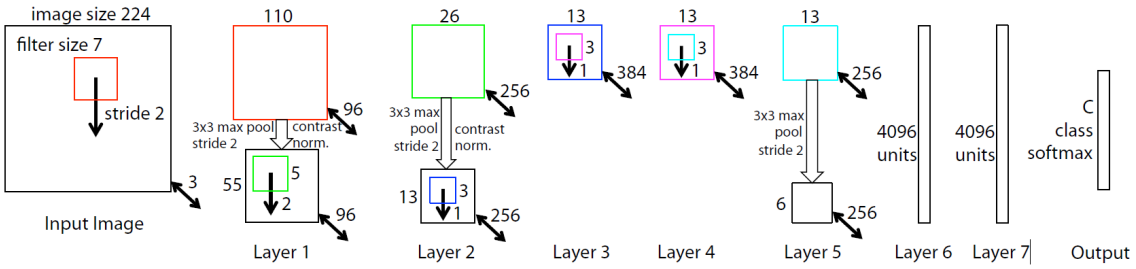


Figure 2.2: Example of architecture of 8 layer CNN model.

It is difficult to explain the motivation of the value y_i trying to'open' the black box,$\hat{f}(x)$. In some supervised machine learning algorithms the outcome can be produced by regression or classification algorithms (e.g. based on decision tree models) exploiting some training parameters . For other algorithms it is not enough to simply look at these parameters to reveal their internal mechanism and explain their prediction. There is a wide variety of black boxes in literature. To cite an example, SVM is a black box classifier that searches for decision boundaries,i.e. optimal separating hyperplanes corresponding with the largest margin. In addition, SVMs are very effective since they can be trained even in case of complex, non-linear decision boundaries. This kind of black box can be, thus, difficult to explain [25]. Random Forest combines predictions of different decision trees, each one trained on an independent subset (with respect to features and records) of the input data. The generalization error of a random forest depends on the strength of the individual trees in the forest and the correlation between them. Using a tree-type form it is usually hard to understand the behavior of the classifier, especially when it has many trees [28]. Some black models are trained into a complex way, thus, an explanation of them is not easy to understand. This concerns, for example, neural networks whose structure can be composed of many hidden layers. Having access to model properties such as weights or structural information to provide explanations is really needed? [23] . In theory, we could more closely consider each individual neuron and activations in the NN but it would be difficult to understand and lengthy. All of this involves not feasible explanations [28]. Convolutional Neural Network (CNN) [31] are mostly used for computer vision image recognition. CNN could be seen as a black box since made by a set of similar layers (an illustration of the architecture is in Figure 2.2) to which they relate on specific properties, like object, shape, contour, color. As an example of a biased decision, it can be seen in Figure 2.3, the image classifier tries to recognize between wolves and husky

(a) Husky classified as wolf (b) Explanation

Figure 2.3: Raw data and explanation of a bad model's prediction in the "Husky vs Wolf" task.

dogs but most of the time we find a mistake when detecting snow on the background. Every time it sees snow, it predicts wolf and if it does not see snow, it predicts husky [26]. We can also have the opposite of a black box that , called a white box, and it is referred as an interpretable model.

If you are building machine learning models, what we are really looking for is a result of the prediction in terms of accuracy and not understand the behavior of itself. In the case concerning interpretability, we should manage it in other ways.

2.2 Interpretable Methods

Interpretable methods are methods that approximate the behavior and predictions of machine learning algorithms, by allowing better understanding to the humans. The intrepretability "is the degree to which a human can understand the cause of a decision" [22]. From these explanations, humans could start to analyze problems concerning the examined context. If the system predicts the correct result, then with interpretable models we could verify if the explanation is locally accurate, more specifically if the explanation emulates the classifiers behavior for a specific data instance. There are many forms of tool of explanation proposed, ranging from decision trees to gradients of neural networks. Most model-agnostic methods work by perturbing the instance, creating a new data-set of new points close to the instance x [26]. The same authors consider that, although function works on a local complexity, the explanation is not able to fit for unseen instance. Starting from this point it is better to focus on a “local region” to learn how to explain the model [27]. Other methods work with similar

approaches, focusing more on the neighborhood of a data point where there is a higher chance that the decision boundary is clear and simple [14]. The explanation that the method provides is a simple if-then statement, called rule. A rule is composed of two parts: antecedent and classification [2]. The rule lists are human interpretable and this is the reason why it is used, as opposed to black box. In general the Model-Agnostic methods have the great advantage of flexibility rather than other explanation methods, limited to specific model classes. The same method of explanation can be used for any type of model.

By using the same classification approach of tabular data, it is possible to identify the location of the object or boundaries as lines, curves and points on the image. The input is multi-dimensional because it is made of several features. A way to show it in the field of images, is to determine which features of the input (pixels) are important for the prediction made by machine learning algorithm. All of these data are useful to recognize the reason concerning correct or wrong predictions. Some methods [26] highlight the area in the image corresponding the features located. In this way is easy to find and understand which features contributed to the final result (misclassification or not). With other methods [1] instead,we can show the feature importance mask as a grayscale image with the brightness corresponding to the importance of the pixel (the dimensions are the same of the original image).

In BB as Convulational Neural Network, some interpretable algorithm tries to replicate the labels, multiplying a pixel matrix and including non-linear activation functions. The difference between prediction and output is calculated in how much the gradient [19] is different in respect to each output. For a number of cycles or epochs all these forward and backwards steps are repeated finding a score. The entire process is called Gradient Descent.
This is one of the many methods already implemented, used with features-scoring [1, 21, 30, 39, 4]. An example of these [15], tries to find the maximal data perturbation that does not change in respect to the model prediction and with 'interpretableCNN' [40]. Method that helps people understand the logics inside a CNN, showing on which patterns the CNN makes the decision.

At the end of this listing of explanation methods, we have to ask ourself obvious questions regarding performance against explainability. It is necessary to lose accuracy in order to gain better interpretability or to obtain more from it without any loss of precision [17]? Is it maybe possible to manipulate trust arbitrarily separately from prediction accuracy [24]? Explainable AI generates approximate simple models and calls them 'explanations' which are able to

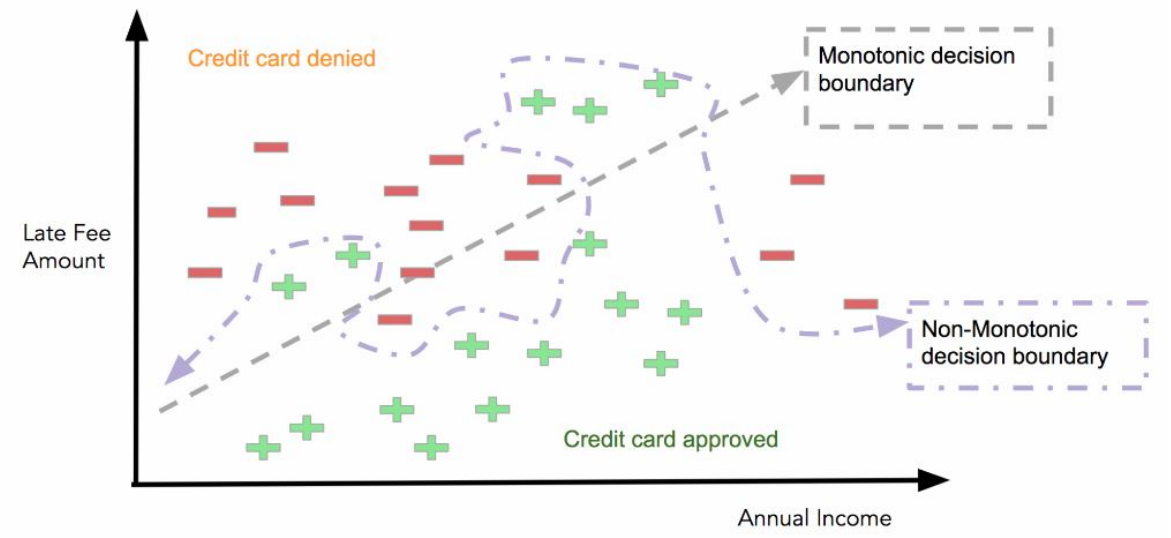


Figure 2.4: Model performance versus interpretability. (Source: *https://www.datascience.com)*

substitute a reliable knowledge of how a complex model works. One often we know that we lose in terms of predictive performance paying a penalty when an interpretable model is chosen but it may be useful compare all of these models to discover a valid hidden answer. But how can we quantify interpretability of the models or what makes explanation human-interpretable [23]? Unlike measures of performance for the machine learning algorithms such as accuracy or F1-Score, these criteria often cannot be completely quantified. There are some properties of explanation methods that help to judge how as good as it may be an explanation. These measures indicate how much the explanations are correcy depending on several factors [29]. The explainability in XAI comes from a set of techniques that add contextual adaptability(local or global) and interpretability,finding an explainability models. The most important property of interpretable models is fidelity. Fidelity is a kind of accuracy measure to achieve how much the interpretable model approximates the black box predictions. The prediction produced by an explanation should agree with the original model as much as possible as:

$$fid_b(M,D) = \text{acc}(y,y') \tag{2.2}$$

The Equation 2.2 measures the fidelity of a black box b using a model M on a dataset D to produce an accuracy between y and y', where y' is found by using the prediction of an interpretable model on a perturbed dataset D'. It is important to highlight that some explanations offer only fidelity on the prediction of an individual data instance or for a subset of the data (perturbation to find new points close to the instance). Another important measure

is to determine if the explanation is comprehensible to the human or not. One example of comprehensibility is the size of the explanation. In addition, can also be considered as comprehensibility concept, if it still possible identify after the transformation made by complex algorithms, the meaning of the original features. From the point of view of developers the amount of time it takes to run methods that generate explanation, called algorithmic complexity, may be regarded as significant. In this work, we deploy a Explainable AI system, where it is possible to compare different behavior of black box with different metrics. After evaluated if the classifier is good or bad, understand which is the reason why the output has been classified in that manner(reliability of rule) and trying to quantify it. After looking at our explanation, we are able to trust or not to trust on the prediction of the black box [11].

2.3 Explainable Artificial Intelligence Tools

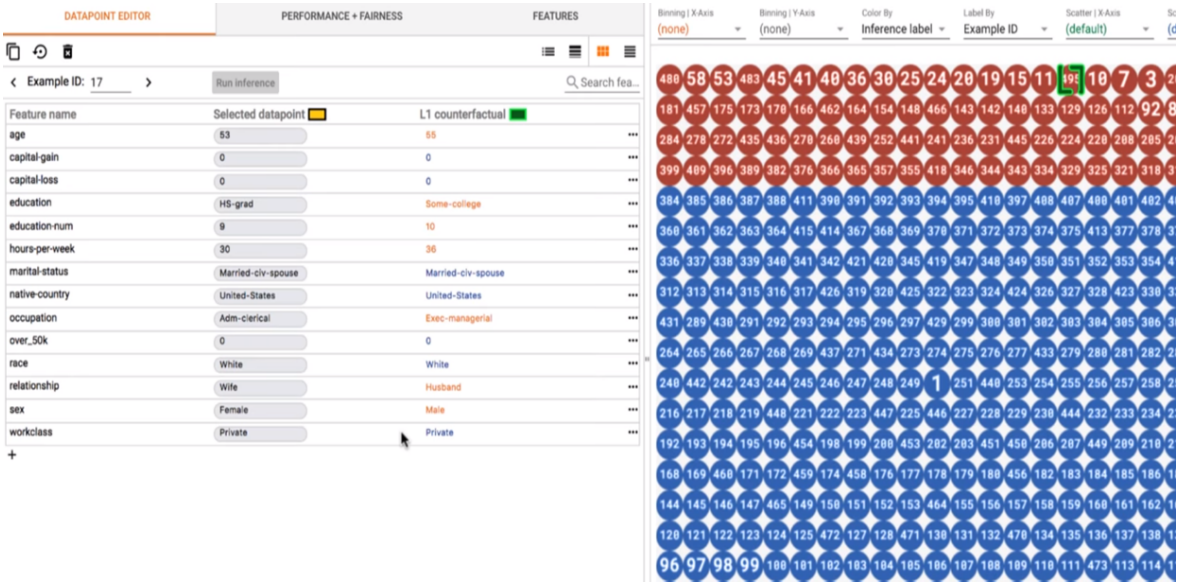


Figure 2.5: Example of What-If Tool which allow users to analyse an ML model without writing a code.

Considerable progress has been gained with regards to the development of new XAI (Explainable Artificial Intelligence) tools. In this section, we describe which are the main tools presented and how can be useful to our goals. As we outlined at the beginning of this chapter, our goal is to create a platform that could help to compare characteristics of a set of explainer,

belonging to a package. There are solutions that works similar, looking and examining the ML model as What-If-Tools [16] [1] given an interactive visual interface to explain some purposes:

1. Compare and experiment on the performance of different models;
2. Inference score of the features, organizing results into confusion matrices, scatterplots or histograms;
3. Visualize distance features between any datapoint;
4. Edit, add or remove features to see the effects on the model;
5. Auto-generated partial dependence plots for individual features;
6. Compare counterfactuals to see what makes the features similar and what makes them different [38].

Going back to the explanation tasks, Skater [7] is a XAI tool for the model interpretation. It is a python project [2] which tries to find a way to evaluate the prediction of a black box by leveraging a combination of existing techniques and different algorithms:

1. *post hoc interpretation*: inspecting the black box or reverse engineering;
2. *natively interpretable models*: the predictive model(explanator function) has a transparent design and is interpretable both globally((as described in section 3.1.1) and locally.

[1] https://pair-code.github.io/what-if-tool/
[2] https://github.com/oracle/Skater

Chapter 3

Explainers

In this chapter, we explain the techniques adopted and implemented in the library, which are provided for a local and global explanation on different type of data.

Type	Framework Explainer	Attribution methods
Tabular	LIME	
	LORE	
	Anchor	
	Corels	
	Skater	Feature importance
		Partial Dependence
Image	LIME	
	DeepExplain	Saliency maps
		Gradient * Input
		Integrated Gradients
		DeepLIFT
		Occlusion
		$\varepsilon - LRP$
	Saliency	Vanilla Gradients
		Guided Backpropogation
		Integrated Gradients
		Occlusion

In particular, we explain the predictions approaches used on tabular (matrix) and on image data.

3.1 Explanation for Tabular Data

3.1.1 Skater

The utility of Skater can be used for the interpretation of global model. As we have already explained in the previous section, Skater is a tool that tries to understand the learned structures of a black box model both globally and locally. It provides a ML system, as in Figure 3.1, that explores the dataset and training both predictive model and interpretable model. Than compares the results trying to interpret and validate the explanations. In the library, it is provided a global model interpretation expressed by statistics and metrics. The result of the model agnostic explanation is in terms of partial dependence and feature importance.

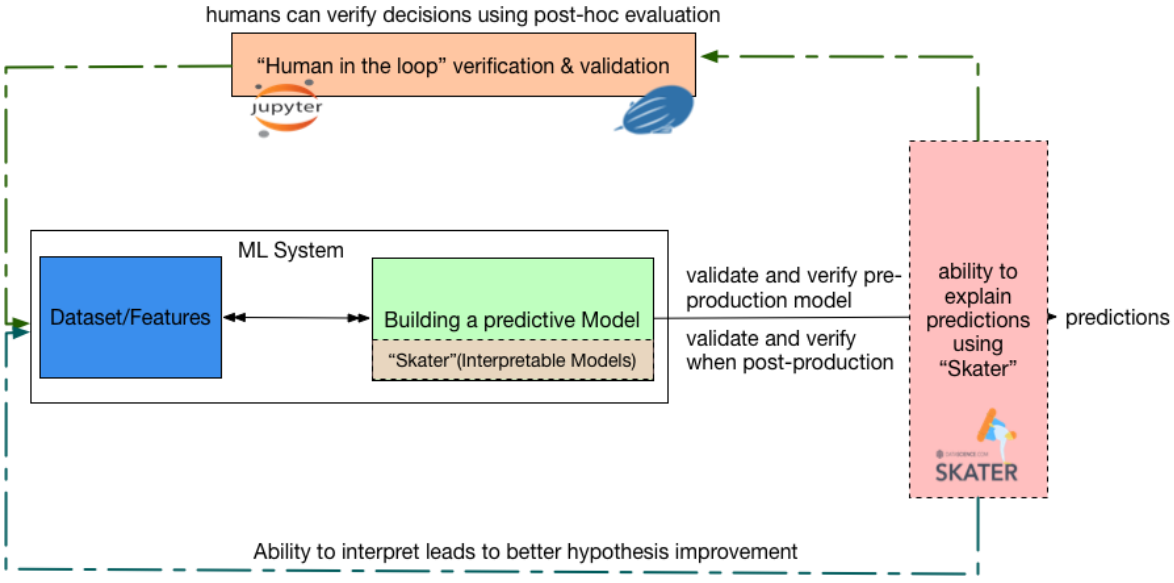


Figure 3.1: HighLevel Design taken on https://github.com/datascienceinc/Skater

Feature importance determines the degree to which the predictive model relies on a particular feature. What is implemented retrieve basically a theoretic information ε, given by a perturbation on a given feature. This information measures the entropy in the change of predictions. If the model's decision criteria depend on a feature, the predictions will change even more as a

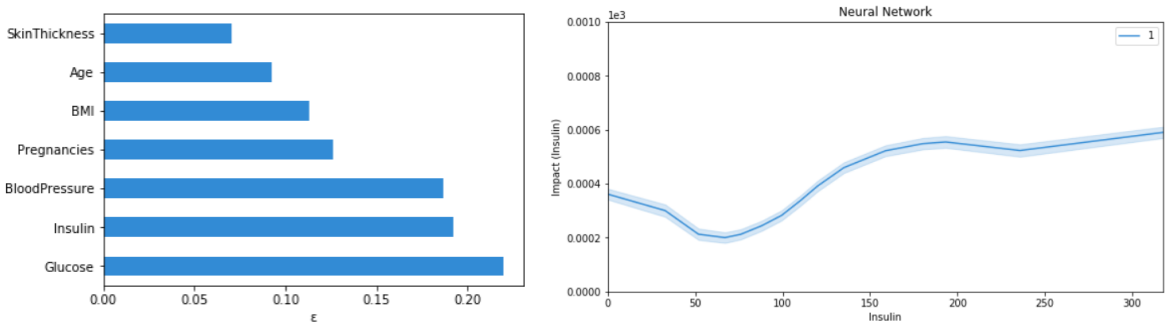


Figure 3.2: On the left the model agnostic feature importance on the prediction using entropy, on the right the marginal effect of a feature on the predicted outcome.

function of perturbing of that feature. "How well does the explanation reflect the importance of features or parts of the explanation?" [23]

Partial Dependence describes, always as a function of its domain, the marginal impact of a feature on model prediction. The derivative of partial dependence describes the impact of a feature (the other features are held constant in the model). The final results show if a feature is linear, monotonous or more complex. In the Figure 3.2 has shown the model global interpretation of Skater.

3.1.2 LIME

LIME (Local Interpretable Model-agnostic Explanations) has been created as a framework of explanations with this aim: "explains the predictions of any classifier in an interpretable and faithful manner, by learning an interpretable model locally varound the prediction" [26]. In order to explain tabular local explanation, LIME generates a new local dataset consisting of permuted samples on which the model use to make the corresponding predictions. LIME produces 'Local Surrogate' models, which is weighted by the proximity of the sampled instances to the instance of interest. It may be expressed mathematically as:

$$exp(x) = \underset{g \in G}{\operatorname{argmin}} \mathscr{L}(f, g, \pi_x) + \Omega(g) \tag{3.1}$$

We interpret our goal as explanation(*exp*) of an instance x .Our model g, that belongs to the set of possible explanation G, tries to minimize the loss $\mathscr{L}$(MSE), which measures how close the explanation is to the prediction of the original model f. π_x establish how much is large the neighborhood around the instance analyzed, this involves a robustness to sampling noise

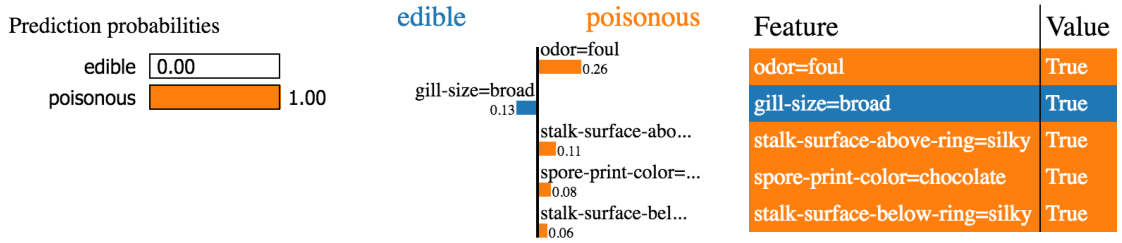


Figure 3.3: An example of the explanation where you can see the probability to be 'editable' or 'poisonous' for such row and how much feature affects (printed by Jupyter Notebook)

since the samples are weighted by it. In addition, we have to consider the model complexity $\Omega(g)$, to keep as low as possible (few features examined).

To define how large the neighborhood is, Lime uses an exponential smoothing kernel. The neighborhood are composed of these new points that are weighted according to the proximity to x. It's really important to find a proper kernel to build the interpretable model with a good proximity measure between x and the others points. Furthermore, the greater is the number of features in the interpretable model, the harder is the interpretation of it. On the other hands, the model produces higher fidelity. The result of this explanation will be an explanation (using a number of features) with the probability of the prediction as in Figure 3.3. [1]

3.1.3 Anchor

Anchor is a kind of explanation technique that wants to explain the behaviour of complex models. It represents local but feasible conditions for predictions. An explanation with *anchor* is: "a rule that sufficiently "anchors" the prediction locally – such that changes to the rest of the feature values of the instance do not matter. In other words, for instances on which the anchor holds, the prediction is (almost) always the same. [27]"

To do this, it uses a sort of 'anchor' on a specific instance. It indicates with a high precision how a model would behave, starting by looking at the 'square' where is the instance as a rule instead of a linear approximation with coefficients(Lime). This square is called "local region" where Anchor learns how to explain the model.

[1] Image contained in: *https://github.com/marcotcr/lime/blob/master/doc/images/tabular.png*

Definition 1 *Let A be a rule acting on such an interpretable representation, such that $A(x)$ returns 1 if all its feature predicates are true for instance x. Let $\mathscr{D}(\cdot|A)$ denote the conditional distribution when the rule A applies. A is an anchor if $A(x) = 1$ and A is a sufficient condition for f(x) with high probability if a sample z from $\mathscr{D}(z|A)$ is likely predicted as Positive (i.e. f(x) = f(z)).*

$$\mathscr{E}_{\mathscr{D}(z|A)}\left[1_{f(x)=f(z)}\right] \geq \tau, A(x) = 1 \tag{3.2}$$

The "local region" mentioned before, that is a better construction of generated data set, has been found selecting the best candidate rule in each iteration. The function 3.2 tries to estimate efficiently the precision of these candidates (*Bottom-up Approach*)[2].

3.1.4 Corels

Corels is a supervised learning algorithm that uses rules lists, also known as decision lists, to obtain an explanation. A dataset is trained of n rule antecedents. Each trained element captures some subset of m samples (or data point) that satisfies the rule's antecedent. Once it is trained, the objective function is defined as: *«the sum of that loss and a regularization term, which is a constant times the length of the rule list.[2]*

$$objective(RL) = \text{los}(RL) + \lambda * \text{len}(RL) \tag{3.3}$$

The loss $\text{los}(RL)$ is the fraction of misclassified samples and λ is a regularization constant that penalizes larger rule lists. It is an accuracy penalty for each rule added to the list. To find an optimal list, the algorithms seeks to minimize the objective function using a discrete optimization technique of branch-and-bound (considering $n!$ possible rule lists). This bounds algorithm allow us to prune as much search space as possible, i.e Corels is efficient when it largely dependent on how much bounds allow us to prune.

```
OPTIMAL RULE LIST
if ({Prior-Crimes>3}) then ({label=1})
else if ({Age=18-22}) then ({label=1})
else ({label=0})
```

Figure 3.4: An example with Corels: rule list comprising of a simply group of rules, followed by a default rule.

[2] To select an anchor with higher coverage, Anchors's library uses Beam-Search Approach.

Corels disposes a Web Application [3] that shows the rule list with the highest accuracy (also called optimal rule list). An example of the type of explanation provided by Corels using rule list is in Figure 3.4. It has not been implemented in the framework for implementation and dependencies issues.

3.1.5 LORE

LORE is a model agnostic algorithm that tries to explain the reasons of the decision taken on a specific instance [14]. Firstly, LORE creates, given an instance x, a neighborhood using a genetic algorithm (Figure3.5). From the balanced set of neighborhood instances previously created, it is defined a local interpretable predictor that produces a local explanation consisting in a decision rule and a set of counterfactual rules. Briefly, it works by the concept of distances between features, whatever they may be (categorical or continuous). In this way it is found a new set of points (neighborhood of records) close to our instance x.

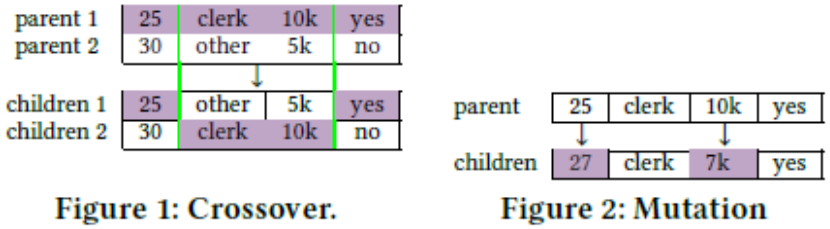


Figure 3.5: Genetic algorithm for "Neighborhood Generation". [14]

At the end is found a rule, using the new set of inputs taking advantage of the interpretable model. The authors choose to use a Decision Tree model, able to give decision rules $r = (p \rightarrow y)$ and counterfactual rules ($\Phi = (q \rightarrow \hat{y})$. Due to the fact that $\hat{y} \neq y$, the explanation is showed as a dictionary with rule and counterfactual rules. The novelty that this algorithm exudes, concerns not only the high expressiveness of the proposed local explanations but also a high-quality training data to learn the local decision tree. Moreover, this framework introduces the concept of counterfactual rules to make interpretability clearer, not present in the other solutions.

[3] A web application of Corels is available in : https://corels.eecs.harvard.edu/corels/run.html.

3.2 Explanation for Images

3.2.1 LIME

For this purpose, LIME is also a model-agnostic that explains the behaviour of classifier that predicts images. It examines the image data by *superpixels*. In this process, it creates a new image as a result of turning superpixel on and off as opposed to perturb on individual pixels that are meaningless taken individually.
In terms of interpretability models instead, uses the same formula seen in 3.1 that choose the model that optimizes the loss part (proximity). To represent image it uses a matrix of pixels and in this case, the explainer demands a three-dimensional tensor as RGB images. Once that has been created the model, it can obtain the image and the mask with the positive weight. This positive weight is the motivation of the association of this area with the prediction of the Black Box, like in Figure 3.6. If *"positive_only"* is equal to False in the function implemented, will be highlighted also the negative weight corresponding to the feature goes against Black Box.

Figure 3.6: An example of the explanation where from an original image it explains why it is a *"tricyle,trike,velocipede"* (printed by Jupyter Notebook)

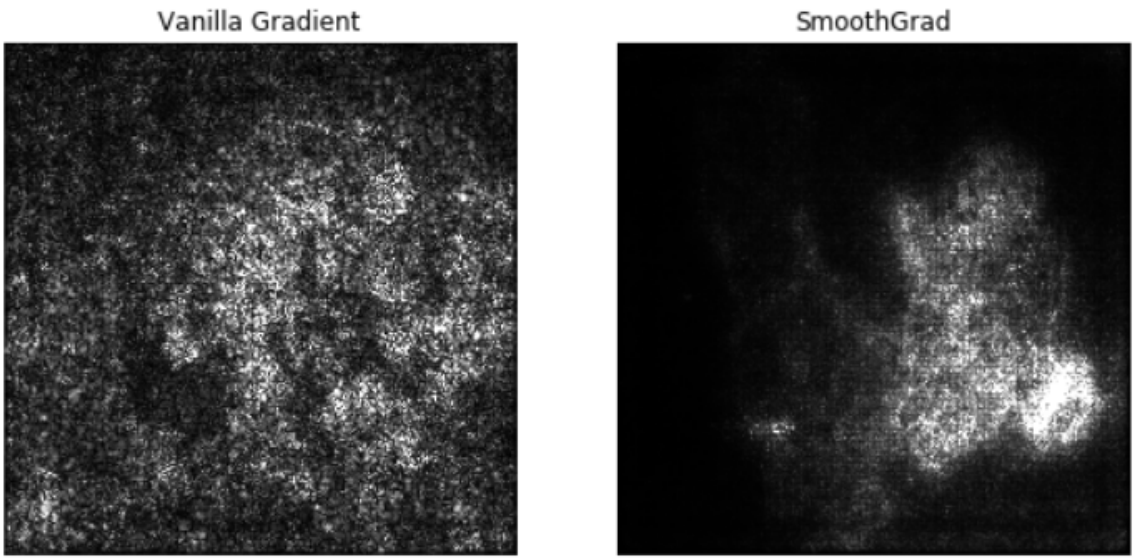


Figure 3.7: An example of the explanation using vanilla gradient method [5].

3.2.2 Saliency

Saliency[4] is a repository of saliency methods that uses mask techniques, enhancing *SmoothGrad* approach, to give an explanation. The results are shown for each method in two ways; normal mask and an image 'smoothed'. SmoothGrad denoises sensitivity mask adding pixel-wise Gaussian noise to many copies of the image. The results are computed simply with the averages of resulting gradients.

1. **Vanilla Gradients**: Estimates saliency masks with a gradients.

$$\theta_j \leftarrow \theta_j + \alpha \frac{\partial}{\partial \theta_j} J(\theta) \tag{3.4}$$

   Equation 3.4 shows the concept of gradient ascent in the input space. Trying to maximize the activation of hidden input, as is reported in [21], a local maximum of that function is found.

2. **Integrated Gradients** : In this case, the method uses:

   - Implementation Invariance of Ingradients;
   - Sensitivity of LRP or DeepLift [33].

[4] All methods are available here : https://github.com/PAIR-code/saliency/tree/master/saliency

It is proposed a black box as deep network presented formally with $F : \mathrm{R}^n \rightarrow [0,1]$. Furthermore, we have an input X and a baseline $\in \mathrm{R}^n$. The goals of this method is to compute the gradients at all points along the path from the baseline $\bar{x}$ to the input. To keep a satisfactory result, applying integrated gradients, is important to select a good baseline (Defaults set to 0). Formally, the gradient computed in each dimension along the path can be expressed as :

$$\frac{\partial F(X)}{\partial x_i} \tag{3.5}$$

As demonstrated, the result satisfies the axioms of completeness and sensitivity.

3. **Guided Backpropagation** : It uses a deconvolution approach [31], learning explanations by neurons in higher layers of a convolutional neural network. Briefly, given an input image, we perform the forward pass to the layer interested, setting to zero all activations except one. At this point to get a reconstruction, it propagates back to the initial step (image).

4. **Occlusion** : This method slides a window over the image and computes how its occlusion affects the class score. The score reveals whether it is positive (decreasing case) or negative (increasing case) evidence for the class. Occluding different portions of the input image with a grey square can outline the output of the classifier. Techniques of unpool, rectify and filter are needed to reconstruct the layers of convnet to permit occlusion [39] but are computationally hard in terms of time.

3.2.3 DeepExplain

DeepExplain is a framework that provides gradient and perturbation-based attribution methods (some has been discussed in Saliency) [33, 31, 21, 39, 1].
The goal of an attribution method is to determine a value $X \in \mathrm{R}$ for each input feature, calculated in relation with the target neuron. Attribution maps (as in the figure 3.8 below) use the colors to give an interpretation of the image. Red colors indicate features that provide positively to the activation of the target output and the blue one show up which features influences suppression on it.

1. **Saliency maps**: As mentioned above, it tries to maximize the activation of gradient in the input space.

2. **Gradient * Input**: calculates scores according to the difference between the activation of each neuron to its "reference activation" [30](*«the reference input is defined according to what is appropriate for the task at hand»*). It is solved the limitation of gradient-based approaches taking advantage of using DeepLift method because the difference between these variables may be non-zero even when the gradient is zero.

$$\sum_{s \in S} C_{sy} = \delta_y \tag{3.6}$$

The Equation 3.6 denotes that summing over all the contributions of neurons in S to y equals to δ_n. It is done for any neuron belonging S, where S is the number of neurons minimally sufficient to compute the activation of y. The resulting is the difference between activation of a neuron and a reference of y.

3. **Integrated Gradients**: As mentioned above, is calculated the average gradient along the path from $\bar{x}$ to x.

4. **Occlusion**: As mentioned above, this is a perturbation-based method.

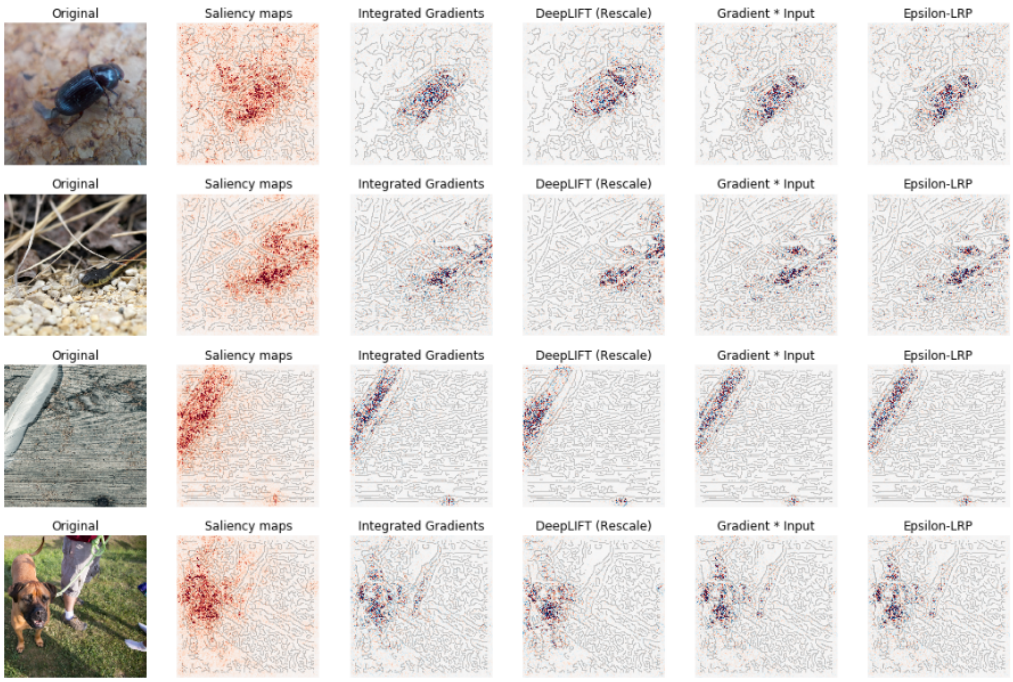


Figure 3.8: An example of the explanation using all the attributions of DeepExplain. [a].

[a] Image contained in: https://github.com/marcoancona/DeepExplain/blob/master/examples/inception_tensorflow.ipynb

5. **DeepLIFT, with Rescale rule and $\varepsilon - \mathbf{LRP}$** : these attribution are implemented as "modified chain-rule". « The chain rule along a single path is the product of the partial derivatives of all linear and nonlinear transformations along the path[1].» It works, if the instant gradient at each nonlinearity is replaced with a function that depends on the method. This method can be implemented as easy as other gradient-based methods known, as stated in the literature review.

All of these methods may give a good explanation depending on the shapes of the image or other factors in evidence. These are for example the absence of strokes where are unnecessary or a heatmap less scattered.

Chapter 4

Explanation Framework

Our work relies on a library that has to guarantee ease and stability to students, researchers, developers or final users that want to understand the behavior of black boxes that treats a large amount of data. There are many programming languages, Python and R to cite some examples, that are more and more commonly used in the field of Big Data Analytics. In fact, any of these guarantee a sophisticated data mining capability and give the full power of flexibility, being able to extend some classes to reach own goals. Python has been used to develop an explanation framework called Xlib, aiming to the evaluation and interpretation of black boxes. Python is a popular open source, object-oriented, interpreted scripting language. It has many advantages such as good flexibility and rich library of methods.The reason for this choice is that object oriented languages easily lend themselves to the creation of extensible frameworks. We can recognize this characteristic of OOP as versatility.

Its versatility has been used to create a framework to evaluate explanations produced by an incrementable number of explainers. The resulting ‘dynamic’ helps us to decide whether if making predictions with that model is reliable and feasible. This versatility ensures the possibility to modify or supply new properties. In fact, with a command line or with a user interface it is possible to report a completely customized data visualization. Moreover is allowed to the user to create/modify sub-commands or a storage backend. In addition, this framework is an API and since it is open, it is possible to create new solutions by leveraging Xlib functionality.

The mindset of the approach used and the methods provided are highlighted in the representation of the conceptual diagram workflow in Figure 4.1. The figure suggests that ‘Explainer’

is an essential ‘bridge’ to utilize different types of data for a different type of black box explanation. This object uses a data model where are collected all the information that are usable for the explanation. With these features is possible to have a benefit from its own contribution without worrying about the structure of the platform. Explainer can evaluate a different type of explanations incrementally. Indeed, each explanation made by methods of explanation stored some properties (and is mandatory to follow this manner) that can be compared with other approaches already implemented. The strength of our work is to provide the possibility to implement its own explanation solutions to evaluate in terms of interpretability.

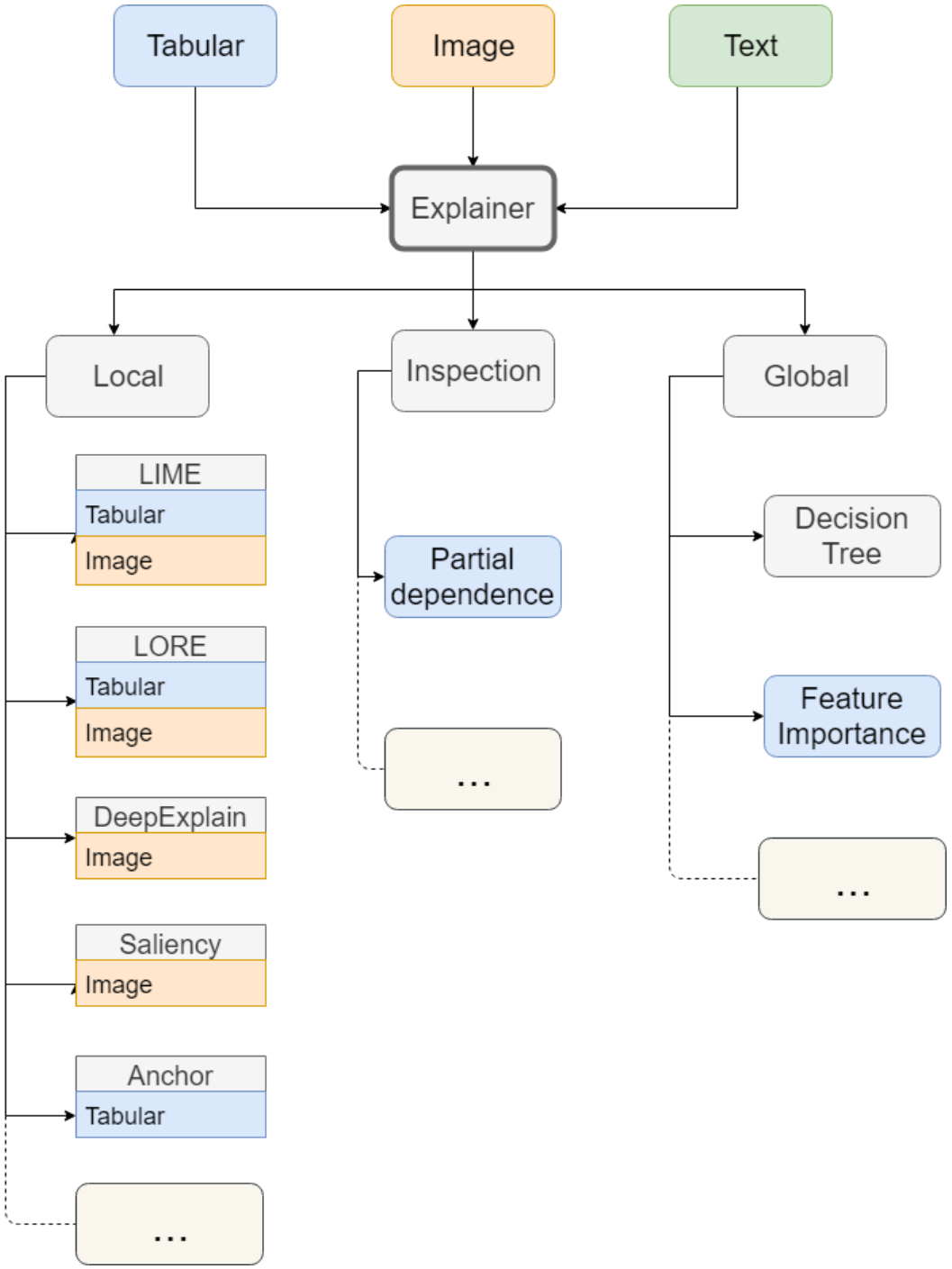


Figure 4.1: Representation of the project listing the categories of black box explanation and its implemented methods [a][b][c][d][e][f].

[a] https://github.com/datascienceinc/Skater/
[b] https://github.com/marcotcr/lime/
[c] https://github.com/riccotti/LORE/
[d] https://github.com/marcotcr/anchor/
[e] https://github.com/marcoancona/DeepExplain/
[f] https://github.com/PAIR-code/saliency/

4.1 Design Choice

XLib[1] adopts object-oriented and functional programming paradigms as deemed necessary to provide extensibility. Xlib consists of 8 files organized into 4 folders with an additional folder as Dataset container as in Figure 4.3.

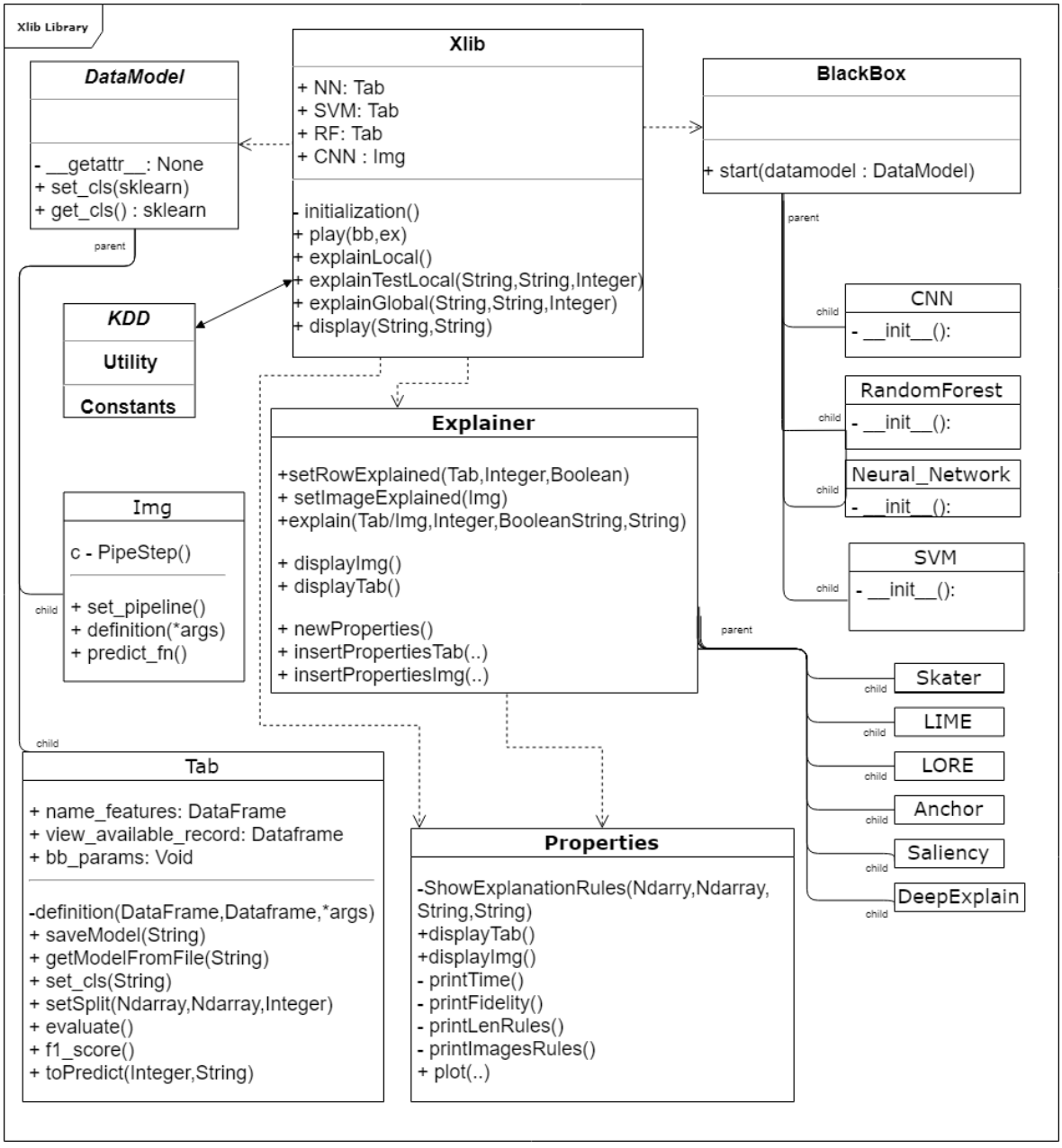


Figure 4.2: UML class diagrams.

[1] https://github.com/Rickyaffo/Xlib

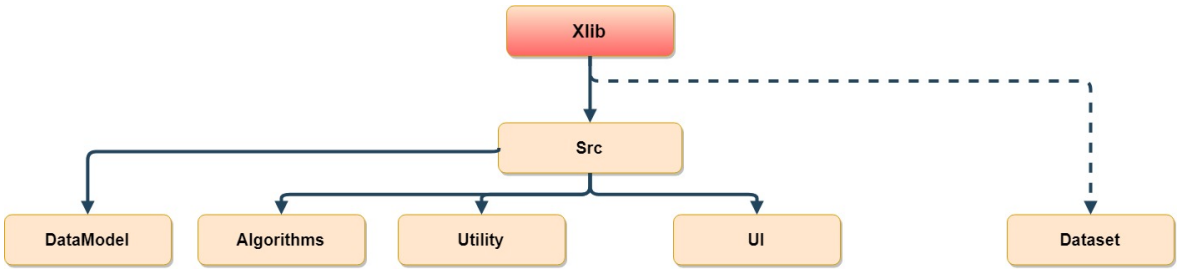


Figure 4.3: Visualization of the package structure in your filesystem

Each files of these are classes or modules used to create, re-use or adapt objects. To reach our goals, Xlib exploits the concept of inheritance. One of the main reasons behind the introduction of inheritance is that it makes the code more easy to re-use. A major benefit of organizing the code into user-defined classes is that similar classes can be reused in the same way in the package. Figure 4.2 describes the structure of the library with an UML diagram by showing the classes, their attributes or methods and the relationships between the modules or other objects. The package contains the class *'Xlib'* that is an abstraction of the examined file. This class is a sort of main which handles the interaction between the user and the tasks of the package.The *'Xlib'* can be identified as the Main Application Class for end users that want either to use the package or to employ it as a test batch that mimics the structure of the class. This main can be replicated as interface or a Shell which imports the *'Xlib'* class. In order to explain all the functionality of the package we will list all the class/modules starting from the main class. For the sake of brevity, in the illustration of platform structure, we leave aside error handling or other functionality as the use of data encapsulation.

'Xlib' contains the following functions and attributes:

1. **XLib**: core class that could be used as interface:

   - *NN - SVM - RF - CNN*: they are @property decorators that get Tab or Img object corresponding with the black box used.
   - *initialization(filename,dataset)*: private method used for ImgFile. It is used to create the dictionary corresponding to black boxes and explainers. Moreover, it sets the default values of the images depending on the origin('dataset') of the dataset.
   - *play(bb,ex)*: For the TabFile, data models are created and then they are inserted in the dictionary 'dictTabModel'. In the ImgFile it is also required the explainer to

create the data properly.

- *explainLocal(ex,bb,row,existing=True,display=False,num_features=5)*: method to call the function 'explain' in the class 'explainer'. In the tabFile, it returns the properties of the explanation. It requiresn an 'explainer', a 'black box' and a row chosen in the test set. If 'existing' has been set to False, it is possible to use a previously initialized array instantiated on the workspace. 'Display' is the function to show the results (default False). 'Num_features' is used in case of LIME explainer. In the ImgFile,if it is needed to the explainer, the attribution method is requested.
- *explainTestLocal(ex,bb,row,existing=True,display=False,num_features=5)*: function used for the experiments that uses 'explain' function to a set of records or images. It returns arrays of 'Property' usable for plotting.
- *explainGlobal(ex,bb,variables*: even this function exploits the function 'explain' but global model explanation. It may add also the number of the column of the variable for partial_dependence evaluation(in skater).
- *display(ex,bb)*:function used for the interaction with the user. Display the explanation, given a name of 'explainer' and a name of a 'black box'.
- *evaluate(bb)*: It is evaluated the black box using the function 'evaluate' and 'f1_score' in the data model.

2. **BlackBox**: class used to work with the black box and save it.
   - *__init__()*: In the initialization, the type of classifier and its parameters are set.
   - *start(datamodel)*: It is added to the data model the classifier that has found through an exhaustive search over specified parameters ('param_grid'). After that it is saved in .pickle or .h5 format.
3. **DataModel**: class where all the features useful to explore the black box and the explainers are encapsulated.
   - *set/get_cls()*: function that sets or gets the black box to the data model object. In the child class Tab or Img, you may override it (Tab case).
   - **Tab**:

  - *name_features*: @property decorator that returns a Dataframe with the names of all the features encoded.
  - *view_available_record*: @property decorator that returns all the records collected in test set.
  - *bb_params*: @property decorator with the task of printing the optimized parameters of the black box.
  - *definition(df,rdf, *args ,target = "class")*: is a sort of __init__, where the 'df' encoded with 'KDD' data processing and the original table 'rdf' are inserted. *args are all the components to encapsulate in the data model object.
  - *saveModel(filename)*: save with the 'dump' function the black box found.
  - *getModelFromFile(filename)*: set the classifier from .pickle file.
  - *setSplit(X,Y,train_size=0.70*: returns a list containing train-test split of inputs X and Y.
  - *evaluate()*: function to evaluate the accuracy of the black box.
  - *f1_score()*: function to evaluate the f1 score of the black box.
  - *toPredict(row, prediction)*: used by explainers to print the predicted target. The number of the row in the test_set and the target are required.

- **Img**:
  - *PipeStep*: Wrapper for turning functions into pipeline transforms (no-fitting).
  - *set_pipeline()*: the building of the black box has to respect some pipeline steps concerning the transformation of the image and to define a classifier (Make Gray,Flatten Image, Normalize, PCA, Random Forest Classifier).
  - *definition(*args)*: defines the object in accordance with the chosen explainer.
  - *predict_fn()*: function to provide the prediction of the black box.

4. **Explainer**: This class supports multiple inheritance and for this reason it is possible to use different type of *'explain'* methods, i.e constructed without altering any of the desirable properties of the workflow(correctness, logic of the task). We have planned a strategy to offer extensibility using the concept of subclasses. The super() function called into our own code is extremely useful when you are concerned about forward compatibility. Here we report the functions of the parent class:

- *explain(model,num_features,display, methodD , methodS)*: This function calls whether or *'explainTab'* or *'explainImg'* depending on the type of data model. The subclass has to implement a function that returns 'Properties' object.
- *setRowExplained(datamodel,row,existing)*: the array examined is inserted in the tabular datamodel .
- *setImageExplained(datamodel)*: it is inserted in the image datamodel, the ndarray with some properties relating to the function of displaying.
- *insertPropertiesTab(rules, num_features, time, c, label, fidelity, target, class_values)*: setter of all the properties requested to store tabular measures.
- *insertPropertiesImg(img2show,mask,title, time, c, label,exp, shape1, shape2)*: setter of all the properties requested to store image measures.
- *displayImg()*: display all the image explanation properties collected in a 'Property' object.
- *displayTab()*: display all the tabular explanation properties collected in a 'Property' object.
- *newProperties()*: each new explanation has to instance a new 'Properties' object.

5. **Properties**: Class that encapsulates the values of the measures and permits their printing.

   - *displayImg()*: call the private function to print the image measures.
   - *displayTab()*:call the private function to print the tabular measures.
   - *printImageRules()*: intermediate function to call 'showExplanationRules(..) properly.
   - *printTime()*: printing time measure.
   - *printFidelity()*: printing fidelity measure.
   - *printLenRules()*: printing length of the rules.
   - *showExplanationRules(imgs2show, masks, title,exp)*: all images are plotted with the used title (original and the name of the explainer) .
   - *plot()*: private function used by 'showExplanationRules'.

6. **KDD**: in this module some functions (‘preparewithNames’,‘prepareImage’) exist to work with the characteristics of the object examinated (more information in sub section). 4.3

7. **Constants**: This module reports all the names that have to be used in the platform. Respectively of the Black Boxes ("RandomForest","Neural_Network", "SVM","CNN"), global tabular explainers ("Skater"), local tabular explainers ("Lime", "Lore", "Anchor"), local image explainers (Lime", "DeepExplainer","Saliency").

8. **Utility**: This module provides all the function that are usable by the interface or needed in the workflow ("getTestSet", "boxPlotExplanation", "open_dataset", "name_image", "getLocallyInstance" ,"getArgs").

This structure supports a Python package dependency considering that we have imported and reimplemented different packages of explanation or other modules shared by the Python community (all the requirements can be found in Appendix B). In addition it is possible to take advantage of the raising of own exceptions, by the Python interpreter, to separate debugging of the package from the side of the developer. There is another package not cited in figure 4.3 above that contains the source code of LORE [2] (LOcal Rule-based Explanations). This code has been modified during the building of the package to understand the behavior of an example of explainer and to make some experiments on it by changing some parameters.

```
class Lore(Explainer):


    def explainTab(self, datamodel,display):
        self.newProperties()   #new ‘Properties’ object
        blackbox = model.get_cls()   #get blackbox fromdatamodel
        def bb_predict(X):  #inner function to predict an instance
            return blackbox.predict(X)
        bb_outcome = bb_predict(self.row.reshape(1, -1))[0]
        stratify = None if isinstance(datamodel.class_name, list) else
            datamodel.rdf[datamodel.class_name].values
        #train model for explainer
        _, K, _, _ = train_test_split(model.
```

[2] https://github.com/riccotti/LORE

```
        rdf[datamodel.real_feature_names].values, model.
        rdf[datamodel.class_name].values, test_size=0.30,
        random_state=datamodel.seed, stratify=stratify)
        t0 = time.clock()          #
        explainer = LOREM(K, bb_predict, datamodel.features_names,
        datamodel.class_name, model.class_values,
        datamodel.numeric_columns,datamodel.e,neigh_type='random',
    categorical_use_prob=True,continuous_fun_estimation=True,
        size=1000, ocr=0.1,multi_label=False,one_vs_rest=False,
        random_state=model.seed,verbose=False, ngen=5)
        try:  #if the explainer doesn't find the rules, use exception
            self.explanation = explainer.explain_instance(self.row,
            num_samples=1000, use_weights=True, metric=neuclidean)
            s = self.explanation.rstr()
            self.properties.CounterFactuals = self.explanation.cstr()
            length = len(self.explanation.rule.premises)
            fidelity = self.explanation.fidelity
            self.properties.CRules = self.explanation.crules
        except ValueError:
            s = "No rule"
            self.properties.CounterFactuals = ""
            length = 0
            fidelity = 0
            self.properties.CRules = []
        t = time.clock() - t0
        #store all the values in the 'Properties' object
        self.insertPropertiesTab(rules=s, num_features=length, time=t,
        c='g', label='Lore',fidelity=fidelity,target=str(datamodel.
        class_values[bb_outcome]),class_values=dict(enumerate
        (datamodel.class_values)))
        if (display): #if it is true, prints the results
            self.display(datamodel)
```

```
43         return self.properties
44     """Function to display the result after 'explainTab(..)' """
45     def display(self, datamodel):
46         print('e = {\n\tr = %s\n\tc = %s    \n}' % (self.properties.
47         Rules, self.properties.CounterFactuals))
48         datamodel.toPredict(self.row, self.properties.target)
49         super().displayTab()
```

Listing 4.1: Example of Lore modified for our purpose. (in appendix C all explainers are available)

4.2 Platform Structure

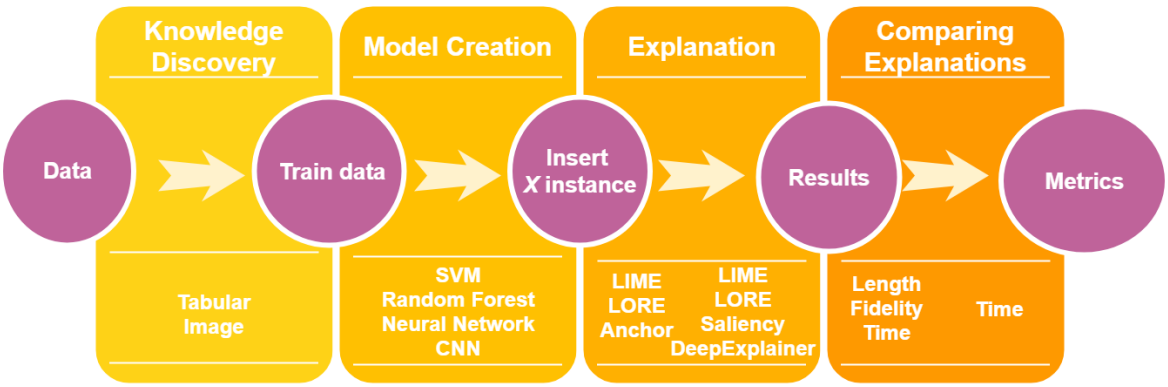


Figure 4.4: All the steps followed by the framework.

This section describes the direction that we have taken to develop all the structure of the package and outline a sort of guidelines to its use. The project has been built following a pipeline idea as in the Figure 4.4. This enabled us to be able to handle different tasks consisting of serial steps, complex dependencies and data file types, fixed and user-defined parameters. The Xlib library aims to meet these needs by:

1. defining core objects for data representation;

2. providing a well-tested set of structure configurable for many applications;

3. establishing an open platform for researchers to collaboratively develop algorithms by principles of explanation or by experiments.

According to the analysis of the process, the platform structure is divided into four phases including Knowledge Discovery layer, Model Creation layer, Model Explanation layer and Comparing Explanation results layer. The first two steps are automated processes defined as ad-hoc tasks, that help to organize inputs in the correct way to prepare them for the 'model explanation' phase. The latter phase is made for establishing relationships among the type of explainer and its dependencies. At the end, a tool has been proposed to show the results and aggregates of certain interpretability measures. The class *'Xlib'* links to other classes or modules that follow the design of the pipeline (Figure 4.4).

Class	**Phases**
KDD	Knowledge Discovery
DataModel	Model Creation - Explanation
BlackBox	Model Creation
Explainer	Model Explanation
Properties	Comparing Explanation results

Table 4.1: List of phases covered by the classes.

In this project, Jupyter[3] has been the open-source software that we have used as a workspace to develop and test the proposed methods. We have used Matplotlib component to draw the results.

4.3 Knowledge Discovery

In the Knowledge Discovery phase, it has been analyzed the type of data used. We want to provide to the user with the functionality of preparing the own data through a standard preprocessing, subsampling and others transformations. In this manner the data can be explored trying to figure out each feature of it. To do this, some functions have been implemented able to cover multiple aspects concerning data exploration, labeling and resizing.

[3]https://jupyter.org/

In this step, user can choose the existing dataset into the local folder [4] [5] or other types of available datasets(mnist) as in the script code below:

```
>>>diabetes_obj = TabFile(filename="diabetes.csv")
or
>>>fileImg = ImgFile(filename="ILSVRC2012_val_00000069.JPEG")
```

In the image case, it is required the name of the file locally stored or the name of the dataset available to download (MNIST or others). If the dataset is stored locally, it is not required to specify the name of it. After choosing the image, it will be converted into a unique format for its proper functioning in the explainer. A function *name_image* has been developed to retrieve all the usable names for the explanation. In the tabular case, there is an encoded CSV(Comma Separated Value) file where at each file correspond its own '.names' file description. We have included this utility to read the type of the data faster and much more efficiently. In the .names file, each column is described in a row with: name of the attribute or class, type declaration of the attribute (string or integer), type of the variable (continuous, discrete). At the end of this process, we are able to analyze the tabular data dividing the data in categorical (use one hot encoding) and numerical and permit to explore it with mapping. For the dataset taken as example, a feature dimension reduction has been performed. 'KDD' class has been used as a lower level aspect of the data processing . It prepares the data using some functionality of scikit-learn and OpenCV related to:

- Rescaling data;
- Normalizing data;
- Fix text ambiguity;
- NaN evaluation with mean or median;
- Size and vision issues.

The Python community provides various solutions for both table and image, such as NumPy (linear algebra), a structured arrays, DataFrame, objects in Pandas, or a wrapper of utility as Tensorflow. N-dimensional data arrays are the most common forms of encountered data,

[4] Images: http://www.image-net.org/
[5] Tabular: https://www.kaggle.com/

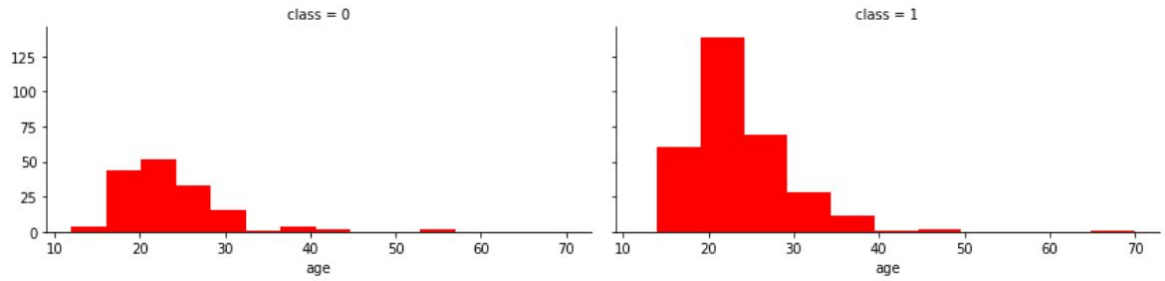


Figure 4.5: An example of the function 'open_dataset' applied for a variable age.

with the task of building the format of the input data. The goal of this step is to gather types of data, standardize it as a unique object and consistently improving it if it is necessary (remove or substitute Nan Values, column exclusion(ID), scale into the unit range allowed). At this point, the KDD function returns an object model(class *'datamodel'* in which the data input (standard format of tabular or image) and many features are encapsulated, useful for operations that we will perform later on. To see the obtained result in the first instance, we provide a function 'open_dataset' (Figure 4.5). With this function, we provide a statistical analysis about how variables in a dataset are related with the total number of the target class. This type of visualization can be useful to see the significance of a variable against another. To get the name of the variables encoded with the preprocessing function,it is just needed the use of a command *name_features* that is a pythonic property of the object.

```
>>>diabetes_obj.SVM.name_features
```

Listing 4.2: List all the name features of the model object.

4.4 Model Creation

Once the object is created, the main class builds a simple model of the black box or restores a previously saved pickle version. The choices of the classifiers available are four:

- Random Forest (*sklearn.ensemble.forest.RandomForestClassifier*)
- Support Vector Machine (*sklearn.svm.classes.SVC*)
- Neural Network (*sklearn.neural_network.multilayer_perceptron.MLPClassifier*)
- Convolutional Neural Network (inception_v3 already present in the folder: *models*)

In the utility module there is a function *getBB(...)*, Listing 4.3,where the object model is obtained with its corresponding selected black box if it already exists in the folder, alternatively the class 'Blackbox' can be used to build it. The .pikle file has to be inserted in the *Dataset* folder and the standard file name must be respected. The library to recognize the black box in .pickle version, shall be defined as follows:"*'nameofthecsvfile'_'nameoftheBB'.pickle*". *BlackBox* class (using multiple inheritance of each family of BB) uses a model selection estimator as GridSearchCV or RandomizedSearchCV to choose the best one(fit and score method).
These are the steps of the function:

1. Model automated testing, validation, and exceptions handling;
2. Hyperparameter optimization, and visualization tooling in general.

The visualization tool produces an evaluation of accuracy and f1_score of the model. At the end of this phase to serialize the ML algorithm into a file we will have, for the dataset managed, the BlackBox saved using pickle operation.
In the case of the images we can build a simple one or use a type of Convolutional Neural Networks called Inception V3 model (introduced beginning with Inception v2 [35], which is an improvement for computational efficiency and low parameter count of Inception [34]). It consists of many convolution and max pooling layers built for the Imagenet Large Visual Recognition Challenge. This kind of black boxes are managed using TF-slim that is a new lightweight high-level API of TensorFlow (tensorflow.contrib.slim)[6] to define or train complex model.

```
def getBB(model,path_file):
"""set the classifier to the model if it already exists,
 otherwise compute a new one."""
    if (Path(path_file).exists()):
        model.set_cls(model.getClsFromFile(path_file))
    else:
        if (Constants.BLACKBOX[0] == model.name):
            m = RandomForest(model.seed)
            model = m.start(model)
```

[6] https://github.com/marcotcr/tf-models/tree/master/slim

```
            if (Constants.BLACKBOX[1] == model.name):
                m = NeuralN(model.seed)
                model = m.start(model)
            if (Constants.BLACKBOX[2] == model.name):
                m = Svm(model.seed)
                model = m.start(model)
            if (Constants.BLACKBOX[3] == model.name):
                 m = CNN(model.seed)
                 model = m.start(model)
        return model
```

Listing 4.3: Set the Black Box to the model.

From this point, we can already talk about the concept of extensibility because if the developer wants to add another black box to examine, it is necessary to add a new condition in the get(bb) function where the new Black Box inserted is called (in addition is necessary to add the name of the Black Box in Constants class).

To summarize these are the few lines to work with the black box:

```
>>>diabetes_obj.play(BB)
>>>diabetes_obj.evaluate(BB)
```

Listing 4.4: Request the Black Box wit the command play(BB) and then evaluate it.

As we will see in the next two sub sections, they likewise lead to the same output but where we have defined different methods of explanation.

4.5 Model Explanation

This subsection covers our main approach, namely to cover a possible growing number of explainers. We have developed this layer with the aim to create a "White box testing" where the researcher may catch defects that are unobserved or that can't be tested directly in the user interface. To get started, the criteria that we have used for the definition of the class *'Explainer'* follows the inheritance principle. Once an explainer is created or instantiated from its default constructor, each object will have custom attributes for its activity. These attributes provide a way to add 'dynamism' to the application and to create an incremental environment well

suited for team development. This approach can be useful to prevent the “feature overload” problem which occurs in case of excessive overload of the parent class. An effect of the "feature overload" problem is the inheritance limitedness which makes the "parametrized" functions totally unmanageable. Thus, to make the problem easier and stable and to guarantee the functionality of the customization, we propose a solution based on kwargs arguments. The *‘Xlib’* component has two functions available: one for local explanation (*explainLocal*) and another for global one (*explainGlobal*).

```
>>>diabetes_tab.explainLocal(ex = "Anchor",bb = "RF",row = 35,display=
   True)
or
>>>diabetes_tab.explainGlobal(ex="Skater", bb = "SVM",variables=4)
```

Listing 4.5: Example usage: command to get different explanation

```
    def explain(self,img,**kwargs):
         #add functionality
        print("Prediction class: " + str(img.prediction_class))
        return super().explain(img,**kwargs)
```

Listing 4.6: Customization usage: Call of super() function in your own Explainer

By adding it to your code and follow the provided documentation, you are ensuring that your work will stay operational into the future with only a few changes across the board. This is your own part where it is possible to insert functionality with references to your own package. In fact, the developers can quickly add or customize behavior by subclassing ‘Explainer’ (see subsection 4.7 for more information). The defined subclass has to override some methods as *explain* with a matching argument signature to a proper use (also in this case it is necessary to add the name of the explainer in the Constants class). In Listing 4.6 it is also showed the possibility to add some functionality of printing before or after the explanation. This function returns the properties necessary for the interpretation layer.

Firstly, we explain how the package of Skater has been implemented, only in the global model interpretation side.

Skater provides two modules:

- *explanations.Interpretation*: It returns Interpretation object with the global interpretations;

- *model.InMemoryModel*: It returns global model using the black box.

To refer to the modules provided by Skater it is necessary to call explaingGlobal(..) from 'Xlib', where it is possible to indicate the type of explainer used, the black box and which is the variable that you examine for the variable dependence (the variable dependence is set to False by default if no value is inserted).

```
    def explainGlobal(self,ex,bb,variables=0):
        model = self.dictTabModel[bb]
        explainer = model.dictTabGlobalExplainer[ex]
        explainer.explain(model,variables)
```

Listing 4.7: Example usage: global explanation

The 'ex' required in the case of "Skater" will provide two results: with the first we provide the weight of the variable of the model(*entropy*) and with the second one we understand the partial dependence, highlighting the relationship between a variable and a model prediction. For the local explanations, the methodology used is the same, changing with explainLocal(..) and adding kwargs keyword. The reason that is behind the keyword with the double star as a parameter allows us to pass the the requested arguments that the explainer needs to use (and any number of them). Additionally it is required the number of the row of the test set (to see the rows available use property *view_available_record*), created in data preparation phase, to elaborate the explanation locally.

```
def explainLocal(self, ex, bb, row, existing=True, display=False,
    num_features=5):
    kwargs = getArgs(None,display=display,num_features=num_features)
    model = self.dictTabModel[bb]
    explainer = model.dictTabLocalExplainer[ex]
    if(explainer.setRowExplained(model, row,existing)):
        explainer.explain(model, **kwargs)
        self.dictTabModel[bb].dictTabLocalExplainer[ex] = explainer
        return explainer.properties
```

Listing 4.8: Example usage: local explanation

In order to uniform the three implementations of tabular explainer, it has been followed what Lime does with the command "explain_instance". First of all, the explainer discovers the

explanation with its own technique and then saves it in an object called ‘Properties’. Lime provides the possibility to evaluate the number of features, *K* corresponding to the length of explanation as you can see in Listing 4.9.

```
>>>diabetes_tab.explainLocal(ex = "Lime",bb = "RandomForest",row = 5,
   display=True,K=3)
```

Listing 4.9: Example usage: LIME explanation

One important thing that we have to highlight is that this tool is not closed only to the single instance of the test-set or to some images only but you can add or modify new instances. To do so, we just need to add the images in the "Dataset/images" folder or pass to the parameter ‘row’ an array with the same length and type requested by the black box.

```
>>>diabetes_tab.explainLocal(ex = "Lore",bb = RF,row = 0,display=True)
    r = { Glucose <= 138.45, Glucose > 120.89, BMI > 28.74,
   BloodPressure <= 79.10 } --> { class: 1 }
    Time spent = 4711 ms
    Length rules: 4
    Fidelity: 0.93
>>>new_record = [0.0, 100,66,40,0.0,34.3,22.0]
>>>diabetes_tab.explainLocal(ex = "Lore",bb = "RandomForest",row =
   new_record,existing=False)
    r = { Glucose <= 120.89 } --> { class: 0 }
    Time spent = 4751 ms
    Length rules: 1
    Fidelity: 0.94
```

Listing 4.10: Example usage: comparison with multiple Lore explanations.

Thankful to this possibility of comparison, is provided a strong evidence between multiple explanations. A strong evidence between multiple explanations is provided in the example above where the value of Glucose has been changed(131.0 to 100.0), knowing its high importance of it on the model. In this manner is easy to see the change of the behaviour of the model (the high importance of it in changing rules, length and fidelity). Examples of explanations output can be found in the next subsection, where we explain how the main interface shows the result to the user.

4.6 Comparing Explanation Results

This layer has been developed aiming the user's perspective but trying to have a look at how particular features have been implemented. In the evaluation of the explanation results, we want to produce a standard answer to the final user. This response has been declared in the function *explain* as an object called Properties, where all the measures of comparison are stored. Each 'explainer' object contains the explainability model and its properties (but it is possible to define other properties that hold their own variable/function for additional purpose). What we are proposing to the user is to have the possibility of comparing the explainers with these measures:

- **Tabular**
  1. Time, measures the time spent by the function 'explainTab' to explain a single record;
  2. Length, represents the length of the rule;
  3. Fidelity, measures how much the explainers approximate the black box predictions.
- **Image**
  1. Time, measures the time spent by the function 'explainImg' to explain a single image.

In this list the measure of coherence is not reported, a measure that estimates the consistency among instances with the same target. Coherence can be evaluated by:

- Jaccard Index: computes the intersection of the elements of x and y divided by the union of x and y. More formally:

  $$D(set(x), set(y)) = |x \cap y| / |x \cup y| \tag{4.1}$$

  J is a statistical index that measures how many rules two explanations have in common out of how many rules they have in total.

- Coefficient D: This similarity measures how much the rules change in terms of length, considering the min/max ratio theoretically ranging from 0 (complete inequality) to 1 (complete equality).

  $$J(set(x), set(y)) = \frac{\min(x,y)}{\max(x,y)} \tag{4.2}$$

This measure can be used only for testing with the function 'explainTestLocal' that iterates using the 'explain' function and append all the properties in an array.

Basically, to show an explanation in a simple and structured standard manner we have to call the function explainLocal(..) and then display(*'bb','model'*). The standard results for each kind of explainer is shown in in the Figure 4.7. The result can be evaluated both individually or comparing multiple instances.

```
>>>a = diabetes_obj.explainLocal(ex = "Lore",bb = RF,row = 12,display=
   False)
>>>b = diabetes_obj.explainLocal(ex = "Lime",bb = RF,row = 12,display=
   False)
>>>c = diabetes_obj.explainLocal(ex = "Anchor",bb = RF,row = 12,display
   =False)
>>>diabetes_obj.display("Lime",RF)
```

Listing 4.11: How to explain a single instance with different explainers(the bb in this case is the same) and display only the LIME results.

These three objects (*a,b,c*) have the measures contained in properties that are compared with the command *boxPlotExplanation("name_measure", **args)*, where it is possible to define the name of the measure following by one or multiple arrays of properties.

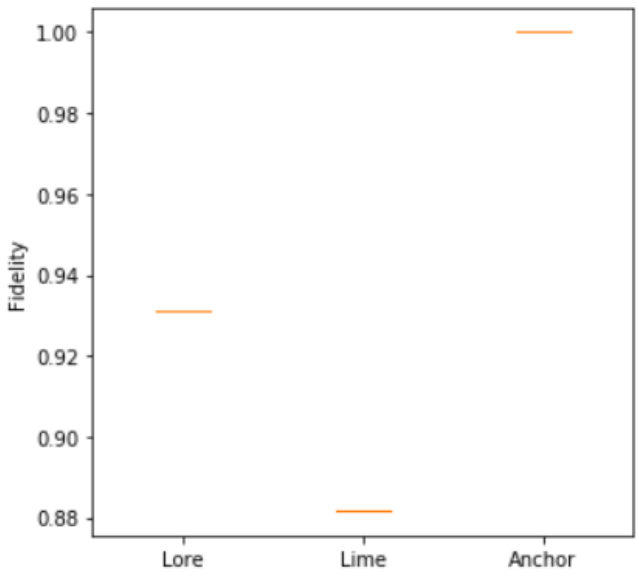


Figure 4.6: How to compare multiple results using BoxPlot in terms of the measure chosen.

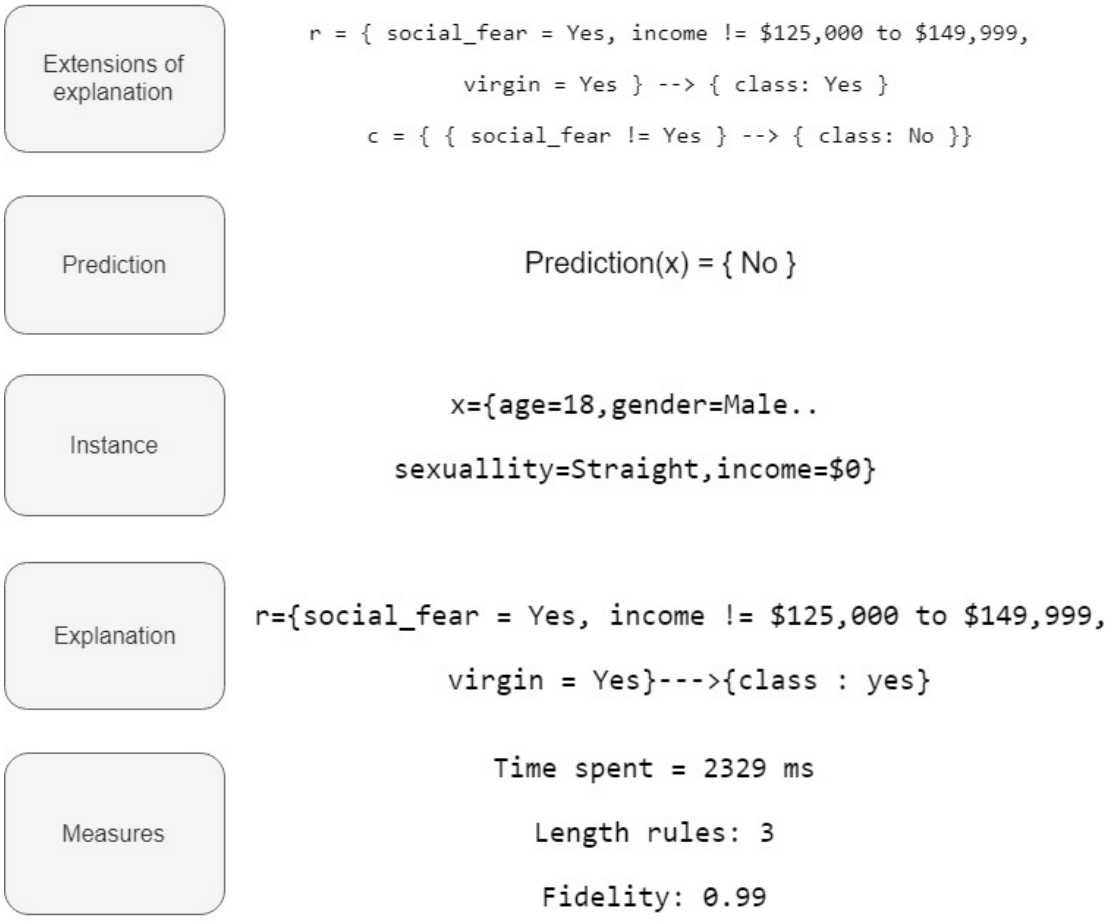


Figure 4.7: If they are present, the function display(..) prints some functionality (1) or information of the method used (Lime shows graphic notebook evaluation), (2) the black box prediction of the model, (3) the record examined, the explanation (4) and its measures(5).

```
>>>boxPlotExplanation("Fidelity",a,b,c)
```

'boxPlotExplanation' is a utility function, independent from the type of black box or explainer. It appends by the name of the measure, all the properties that will be shown using a matplotlib boxplot. In the case of coherence some function are called with the task of calculating it.

4.7 Extending the Library

Xlib implementations support the modular extension of subclasses. Given that doing subclassing guarantees us the flexibility of using the framework, we show how it is simple to do it. We need to create in the 'Explainer' or in 'BlackBox' class, a sub class where it is possible

to simply define methods with the same name of a method in its parent class (concept of overriding). Instead of changing the method based for explanation, it is possible to define same functions in just declared derived class and change the codes.

An example is available in Listing 4.12:

```
class Ext(Explainer):

    def explain(self,model,**kwargs):
        print("Your own Explainer")
        # return super().explain(img,**kwargs)
    def displayImg(self):
        pass
    def displayTab(self):
        pass

class AdaBoost(BlackBox):

    def __init__(self, seed):
        self.param_grid  = {
              "base_estimator__criterion" : ["gini", "entropy"],
              "base_estimator__splitter" :   ["best", "random"],
              "n_estimators": [none, 1, 2]
             }
        DT = DecisionTreeClassifier(random_state = seed, max_features =
    "auto", class_weight = "auto",max_depth = None)
        self.bb = AdaBoostClassifier(base_estimator = DT)
    def evaluate(self,..):
        pass
```

Listing 4.12: An example of how to extend parent class in the library.

A mandatory action is to fill all the fields in the ‘Properties’ class to the correct visualization of the standard output. The parent class function will be overridden by the derived class function. In this manner is possible to create our explanation solutions or distinguish types of

black boxes for the Explainers available.

```
self.insertPropertiesImg(img2show = img2show,mask = mask,title=title,
    time = t,c = 'r',exp = "Ext" ,label = "Ext",shape1 = img.SHAPE1 ,
    shape2 = img.SHAPE2)
```

Listing 4.13: After completing the algorithm of explanation it is required to fill necessary fields to produce interpretability.

Python brings the benefits of a high-level scripting language and for this reason the strength of an object-oriented language is to feel free to extend your own class with other methods even just for testing. In'Xlib' class we have some examples of this use indicating how Python implicitly passes the object to method calls and can use different or common functions.

```
dictTabModel[bb].f1_score()
or
explainer.explain(img,**kwargs)
```

Chapter 5

The framework at work

The goal of these experiments is to test all the explainers specified in chapter 3 for each dataset. The results that we want to discover with these experiments are:

1. How does the fidelity, time and length change by distinguishing the black boxes in tabular case?

2. How well the explanation approaches could be compared between them, when using the interface?

3. How much variation is there in terms of time when an image is explained?

In each section, we present the type of dataset used for the evaluation of explanations, which black box has been computed for this type of prediction and which are the final results from which we can draw conclusions. The results that we have achieved during our experiments use the function *explainTestLocal*. In the function we have created a new series of instances, by standardizing, in the tabular case, the percentage of existing unique classes to avoid unbalanced set (we explore the values from the test-set). In order to evaluate the experiments, we will propose the measures of explainability proposed in section 2.2, considering mean and standard deviation. Moreover for tabular cases we will evaluate also Coherence, another measure of evaluation in terms of consistency of the explanation which measures the relationship between couples of data with the same target. The following experiments were executed on a commodity laptop: an asus VivoBook with an i7-8550U 1.99 GHz processor and 16GB of RAM.

5.1 Experimental Setup

5.1.1 Tabular

To experimentally verify the measures of interpretability, we apply Lime, Lore and Anchor to a list of dataset taken from some repository. We have analyzed *compas* taken from UCI[1] repository and three other datataset as *foreveralone*, *diabetes* and *adult* found on Kaggle[2] community repository. These datasets have been chosen for their variety in terms of size and type of the features. In this manner, we are able to verify the behavior of the explainer in all respects. The characteristics of the tabular datasets are shown in Table 5.1.

Name Dataset	Records	Features	Categorical	Numerical	Class
ForeverAlone	469	15	14	1	*attempt_suicide*
Diabetes	768	8	0	8	*outcome*
Compass	7214	11	4	7	*recidive*
Adult	32560	9	5	4	*salary*

Table 5.1: List of the tabular dataset used for experiments.

5.1.2 Image

ImageNet[3] is an image database of different subject and MNIST [4] is a database of handwritten digits. The first source has to be inserted in the folder '*Dataset/images/*' in order to be processed, instead the other source has been automatically downloaded with sklearn dataset. The characteristics of the images datasets are shown in Table 5.2.

Name Dataset	Number of Classes	Sizes
MNIST	10	$28x28(gray)$
ImageNet	1000	$299X299(color)$

Table 5.2: List of the image dataset used for experiments.

[1] https://archive.ics.uci.edu/ml/index.php
[2] https://www.kaggle.com/
[3] http://www.image-net.org/
[4] http://yann.lecun.com/exdb/mnist/

5.1.3 Black Box

Our goal is not to get better performances of our classifier, by maximizing its precision or other measures of evaluation. Instead, the purpose is to get an explainer that is able to mimic the behavior of variously performing black boxes. We briefly summarize the presented models of the black box in Table 5.3.

Dataset Name	Black Box	Accuracy
Foreveralone		
	SVM	0.66
	RF	0.73
	NN	0.73
Diabetes		
	SVM	0.64
	RF	0.74
	NN	0.67
Compass		
	SVM	0.59
	RF	0.62
	NN	0.62
Adult		
	SVM	0.83
	RF	0.73
	NN	0.73
IMAGENET		
	$Inception(v.3)$	0.78
MNIST		
	$Inception(v.3)$	0.93

Table 5.3: For each dataset file we have listed the corresponding black box model type and the accuracy of a test-set.

See Appendix A for more details regarding the confusion matrix.

5.2 Result

After preparing the set of instances for which we do our experiment, we store for each row the explanation in an array, necessary for the final evaluation.

```
1    experiment_result = []
2    explainer = model.dictTabLocalExplainer[ex] #or  explainer = self.
   dictImgExplainer[ex]
3    for row in test_set:
4        explainer.setRowExplained(model, row)
5        experiment_result.append(explainer.explain(model, **kwargs))
```

Listing 5.1: Script of function explainTestLocal(...) where for each row in the test set(tabular and images) has been found the explanation.

All of these measures are evaluated by *"boxPlotExplanation"*, that is able to evaluate the survey for each arrays of properties. The below table reports the results by Explainer.

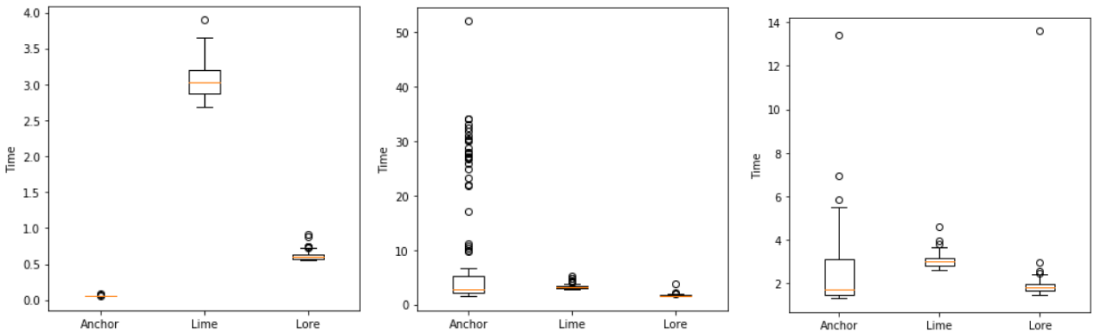
5.2.1 Tabular Explanation

For tabular data experiment, we compare any dataset for all the explainers and all the blackbox:

	Foreveralone			**Adult**		
	LIME	Anchor	LORE	LIME	Anchor	LORE
SVM	3.05±.22	.056±.004	.62±.06	16±6	17±27	11.3±1.3
NN	3.04±.29	2.5±1.5	1.9±1.0	1.95±.26	1.6±1.2	10.6±1.3
RF	3.2±.4	8±10	.68±.23	2.24±.23	8±15	15±7

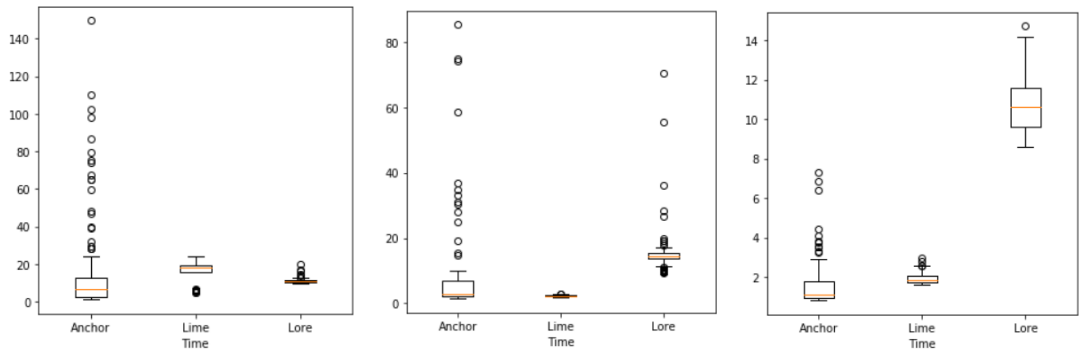
Table 5.4: Comparison of time in **Foreveralone** and **Adult** dataset

With two datasets with much different sizes, we can see that when the dataset is small, Lime generally spends more time for the explanation despite even Anchor for Random Forest is the highest, due to an unfocused and fragmentes distribution. When the dataset is big instead, in the case of SVM the time of all three explainers increases, although Lore is, however, the best one. In terms of standard deviation, the worst is Anchor, for some instance it is three times higher.



(a) Support Vector Machine (b) Random Forest (c) Neural Network

Table 5.5: Box Plot of time scores for the (**ForeverAlone**) dataset



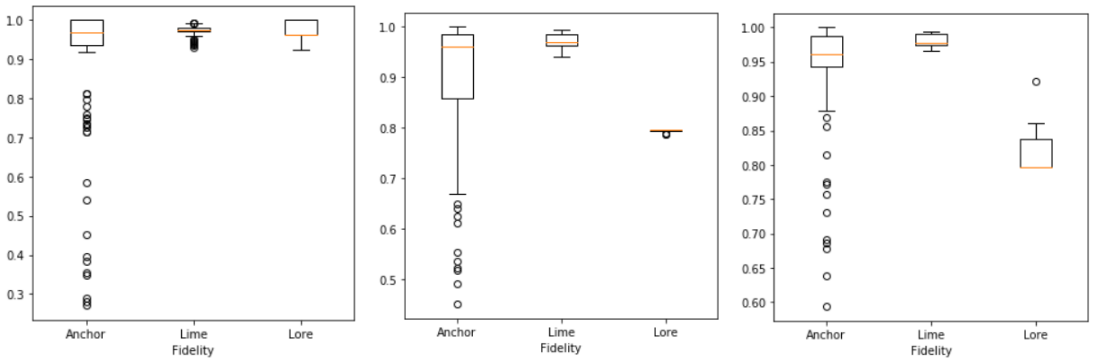
(a) Support Vector Machine (b) Random Forest (c) Neural Network

Table 5.6: Box Plot of time scores for the (**Adult**) dataset

The accuracy of the classifiers (as we saw in the last section) is not high in the case of compas. Nonetheless, fidelity corresponds to the fairness of the behaviour of the black box and the measure calculated is high for each of them. We can highlight furthermore that LIME is stable for each type of black box and each dataset.

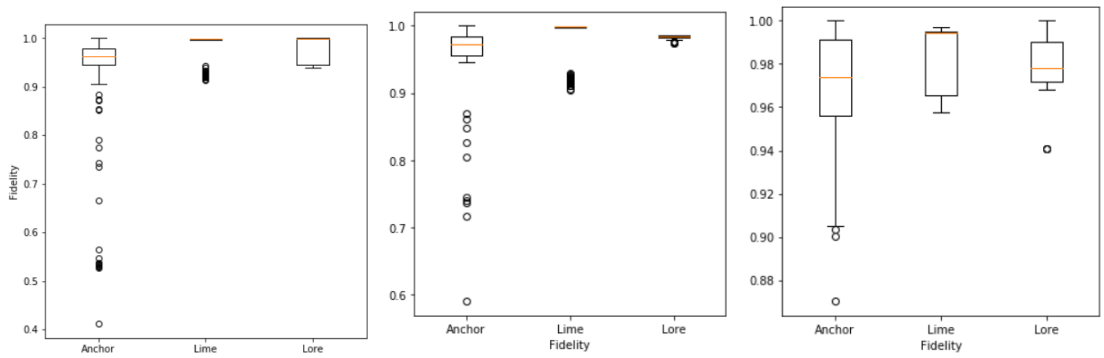
	Compas			**Adult**		
	LIME	Anchor	LORE	LIME	Anchor	LORE
SVM	.974±.011	.901±.185	.977±.024	.982±.031	.927±.12	.978±.027
NN	.980±.008	.943±.082	.816±.023	.986±.014	.971±.024	.981±.011
RF	.973±.013	.908±.144	.794±.001	.981±.034	.962±.06	.982±.004

Table 5.7: Comparison of Fidelity in **Compas** and **Adult** dataset



(a) Support Vector Machine (b) Random Forest (c) Neural Network

Table 5.8: Box Plot of fidelity scores for the (**Compas**) dataset



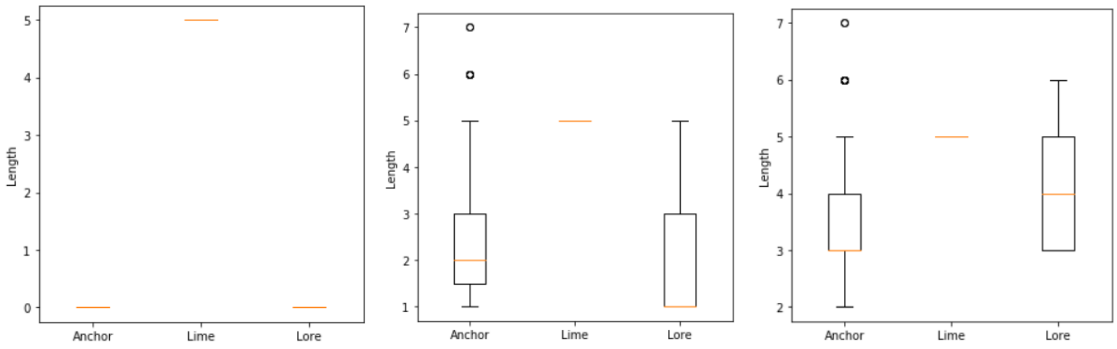
(a) Support Vector Machine (b) Random Forest (c) Neural Network

Table 5.9: Box Plot of fidelity scores for the (**Adult**) dataset

In the case of the length we have analyzed '*Diabetes*' case where for SVM in Anchor and Lore fails to find an explanation, and hence the length is 0 (also the fidelity). In the other black boxes instead, they found an explanation with a length minor of Lime. These examples are useful to compare the difference between rule-based and linear, explanations.

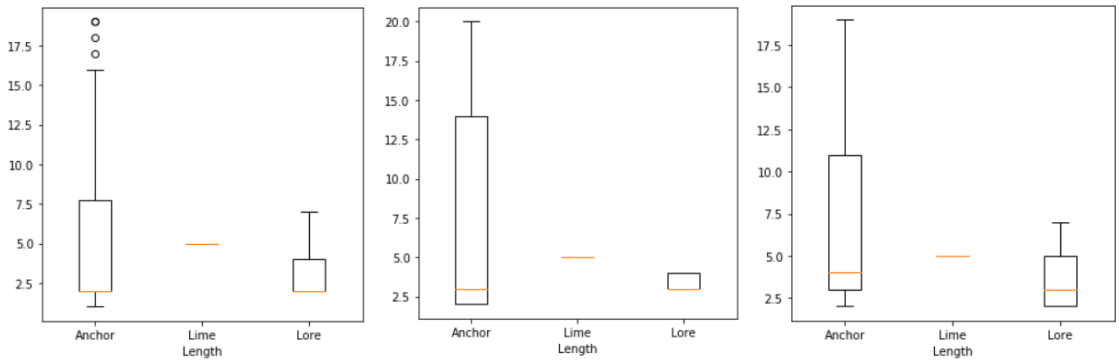
	Diabetes			**Compas**		
	LIME	Anchor	LORE	LIME	Anchor	LORE
SVM	5.0	0.0	0.0	5.0	5±5	2.9±1.3
NN	5.0	3.4±1.2	4.1±1.1	5.0	6±5	3.4±1.4
RF	5.0	2.3±1.3	2.0±1.3	5.0	8±6	3.4±0.5

Table 5.10: Comparison of Length in **Diabetes** and **Compas**



(a) Support Vector Machine (b) Random Forest (c) Neural Network

Table 5.11: Box Plot of length scores for the (**Diabetes**) dataset



(a) Support Vector Machine (b) Random Forest (c) Neural Network

Table 5.12: Box Plot of length scores for the (**Compas**) dataset

With Compas, Lore is able to have better clarity against the other ones (Anchor has a high standard deviation). To see better how the behavior of rules is, we have analyzed the coherence of it.

	Diabetes			**Compas**		
	LIME	Anchor	LORE	LIME	Anchor	LORE
SVM	1	0	0	1	.78±.09	.18±.14
NN	1	.78±.07	.77±.04	1	.61±.06	.74±.04
RF	1	.74±.08	.68±.17	1	.62±.17	.87±.01

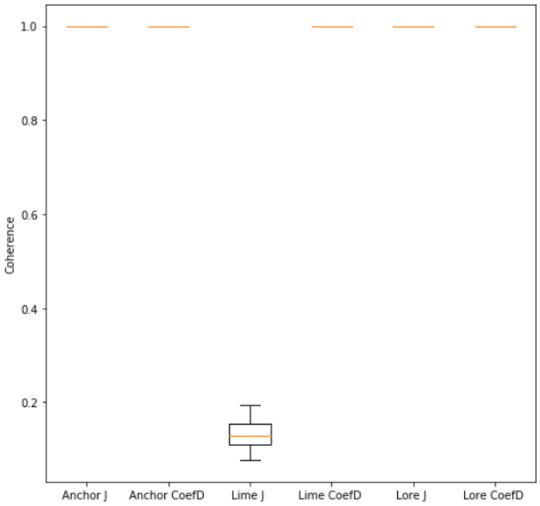
Table 5.13: Comparison of Coef D in **Diabetes** and **Compas**

In the first instance, we see how much is the distance between the rules with the same target, in terms of the length(*min(x)/max(y)* ratio). If we know that it is not possible to measure it in Lime context (the rules are fixed with a parameter), we can see that for the other explainers there is not too much dispersion to find the same target. Lore with Compas has the highest value in Random Forest and Neural Network but defects for SVM. Defect in terms of differentiation of reading of the rules but not on the equality of it. For this reason we have to look also to the Jaccard index.

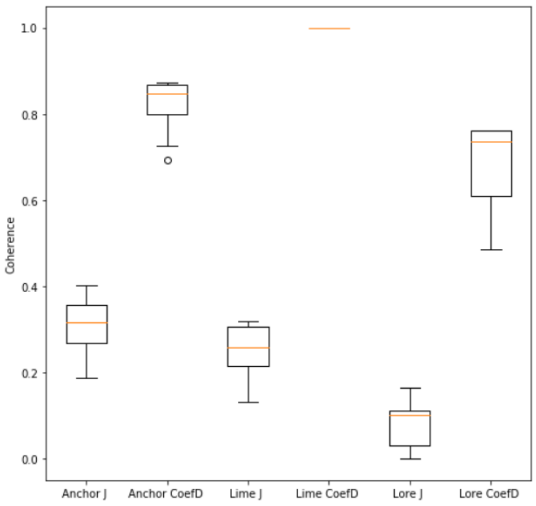
	Diabetes			**Compas**		
	LIME	Anchor	LORE	LIME	Anchor	LORE
SVM	.13±0.03	0	0	.34±.06	.29±.04	.76±.08
NN	.17±.03	.17±.07	.26±.11	.36±.07	.22±.05	.11±.06
RF	.15±.02	.21±.11	.53±.28	.41±.08	.24±.05	.24±.05

Table 5.14: Comparison of Jaccard Index in **Diabetes** and **Compas**

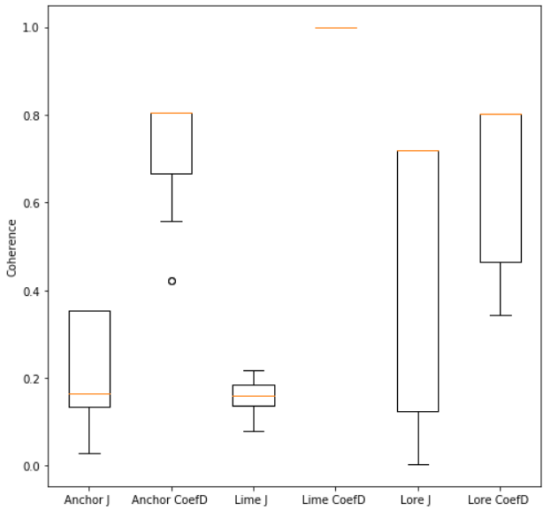
This index is very important since if two set of rules share all members of the rule it means that they are perfectly similar. The closer to 1, the more similarity on average of all the rules. If they share no members, they are not similar. We see that in general Lore is much cohesive than the others. Anchor is close to 0.25 but we cannot define it as the worst considering that the variance of the length was high but with a discrete coherence in terms of coefficient D. Lime takes advantage of its fixed length, and is difficult to compare it with the other measures. With this test, we have seen that jaccard index is a strong measure to see the cohesion of a set of rules made by an explainer, although the complexity of the rules are not able to be taken into consideration in total.



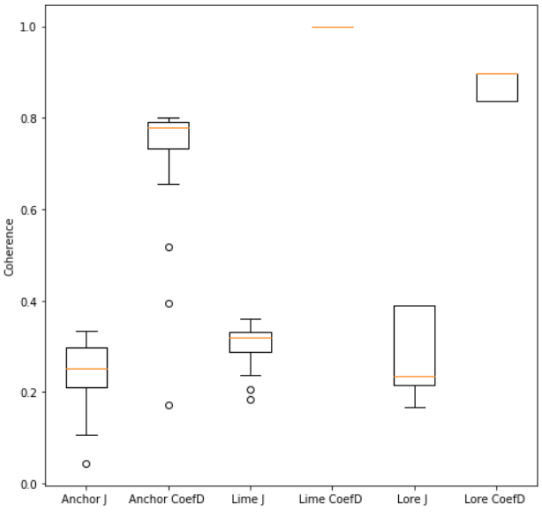
(a) SVM diabetes



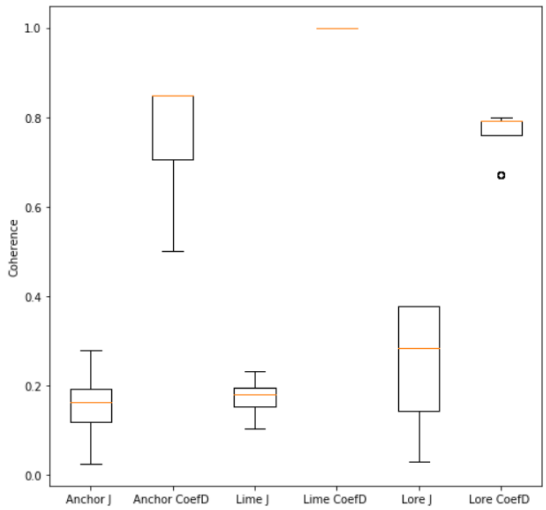
(b) SVM compas



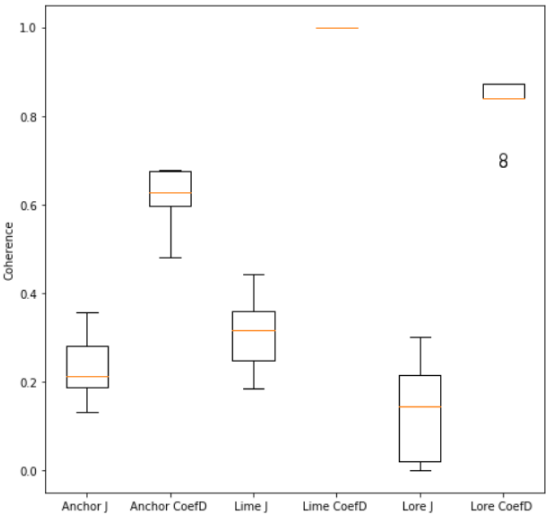
(c) Random Forest diabetes



(d) Random Forest compas



(e) Neural Network diabetes



(f) Neural Network compas

Table 5.15: Box Plot of cohrence scores for (**Diabetes**) and (**Compas**) dataset

5.2.2 Image Explanation

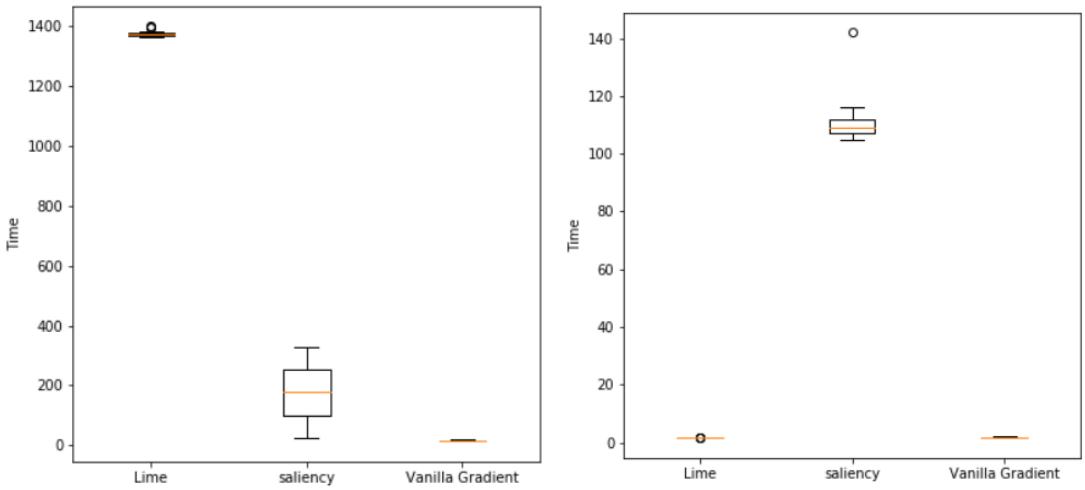
For the Images experiment, we compare the time spent by the algorithms to find the explanation in 100 images with different characteristics. In this experiments for reasons of high time complexity we have excluded the perturbation-based attribution methods of DeepExplain and Saliency.

Time

	Methods	Imagenet	MNIST
Lime	/	1380±9	1.76±0.11
DeepExplainer	grad*input	$(3.3\pm1.9)\cdot10^2$	99.8±1.3
	saliency	$(2.7\pm1.8)\cdot10^2$	109.4±2.1
	intgrad	$(1.1\pm1.4)\cdot10^3$	115.2±2.5
	elrp	$(4.7\pm0.4)\cdot10^2$	122.7±2.5
Saliency	gradient	16.5±1.5	.06±.02
	guided backprop	21.4±1.3	.34±.02
	integrated gradients	$(1\pm4)\cdot10^3$	.06±.01

Table 5.16: Comparison of time in **Imagenet** and **MNIST**

Lime and the other two explainers use different approaches to draw the explanation and the gap between the two scores is understandable and justifiable. The contrast between the time scores is mainly due to the difference in the size and shape of the images. MNIST has a grey input dimension of 28*28 and for some methods based on the gradient of each pixel, is really fast to notice the variation of it. In the MNIST case, we can also observe a linear decrease of the time needed with the exception of intgrad's method increasing slightly in respect to the others.



(a) ImageNet

(b) MNIST

Table 5.17: Box Plot of running time of any of the available explainers.

Chapter 6

Conclusion

The purpose of this study is to propose a library which may be useful to provide a uniform approach to the concept of explanation. This could potentially lead to the evaluation of multiple types of explanation in numerous applications. Moreover, Xlib offers the possibility to examine and develop novel explanation methods applicable to different type of black boxes. In particular it has been created a framework of multiple explainers exploiting the concept of extensibility. Extensibility is a well-known technique used to allow researchers and practitioners to develop explanation methods. An explainable system exploiting the concept of extensibility can have several advantages: (i) first of all, its usage does not require any technical background, (ii) it provides the possibility to explain instances already analysed or to create new ones, and (iii) it is considered very usable (from the user perspective). As a design goal, the utility of incremental explanations largely grows when moving from the success of the design and implementation of inheritance. Indeed, the developer can use this concept to override methods and add own algorithms.

The described tool was tested through diverse input data so as to verify the stability of the platform (i.e. when dealing with different type of data, black boxed and explainers). This library was developed following a top-down approach - starting from a simplified design - in order to guarantee the repeatability and reusability of processes as well as the rapidity of produced results. Future works will go in the direction of making the system more usable through a bottom-up reverse engineering approach. Indeed, a bottom-up approach can be needed - especially when the library produces many solutions - to discover errors that are hidden in the model. Finally, when dealing with images, the experiments reported different

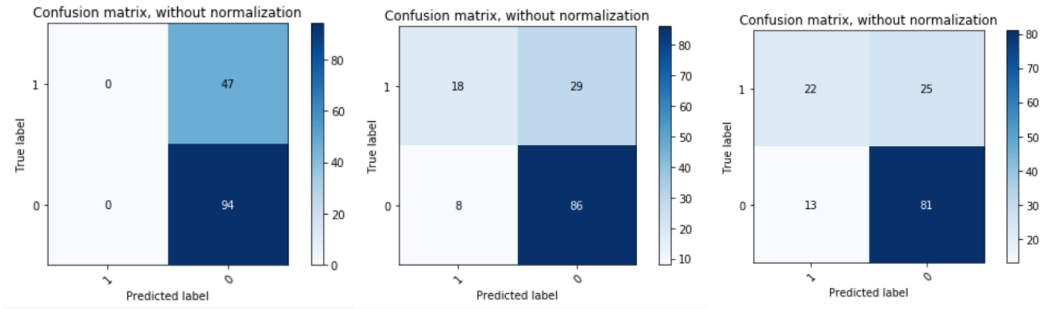
solutions in terms of time complexity, due to the different shapes and sizes of the image. Future versions of this library are also needed to determine a measure similar to the coherence in the tabular examples. This measure can be found comparing how much change the explanation with respect to multiple explanations with the same target. It is possible computing the ratio between the difference among original images and the difference among the masks created with the explanations (using norms for this purpose).

In order to exploit and gain the best from the functionalities of the framework, a developer should use the following three steps : Firstly, verify the trust of the decision system compared with our intuitions and exploring the data. Then understand what our model tells us, in term of behaviour of any data, minimizing or understanding how to fix the error. The comparison between explainers allows the user to have an advantage from the strengths and offset the drawbacks by each of them.

To extend our research we could considered the meta-analytic study of the data as an additional challenging area in the field of machine learning. The meta-analysis [36] is the process to the aggregation of weighted information from multiple similar studies in order to identify the reason for the variation. With this package it could be possible to analyse different sources of these studies and investigate the concept of interpretability.

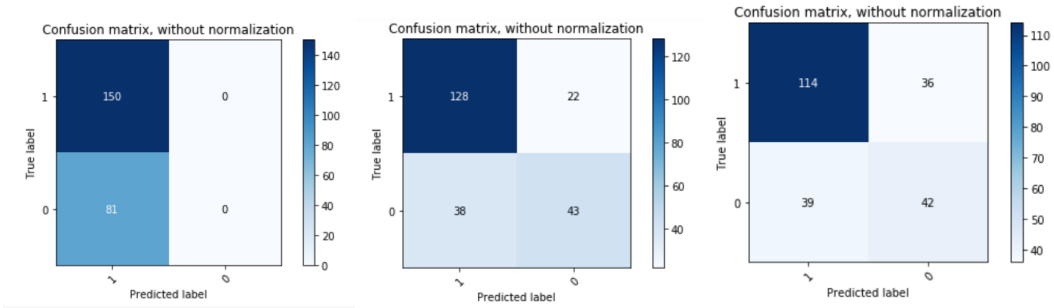
Appendix A

Black Box Scores



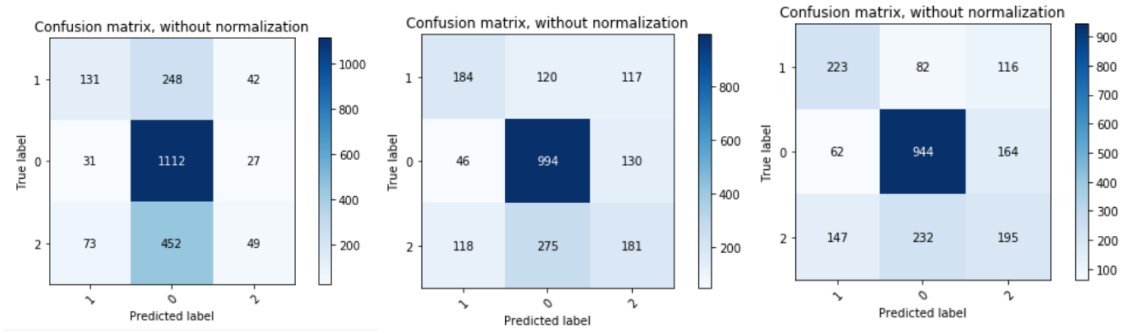
(a) Support Vector Machine (b) Random Forest (c) Neural Network

Table A.1: Confusion matrix: accuracy scores for the (**ForeverAlone**) dataset.



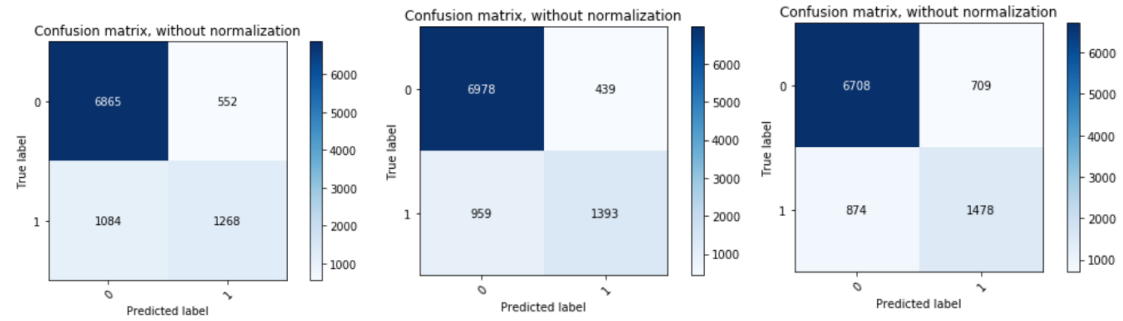
(a) Support Vector Machine (b) Random Forest (c) Neural Network

Table A.2: Confusion matrix: accuracy scores for the (**Diabetes**) dataset.



(a) Support Vector Machine (b) Random Forest (c) Neural Network

Table A.3: Confusion matrix: accuracy scores for the (**Compas**) dataset.



(a) Support Vector Machine (b) Random Forest (c) Neural Network

Table A.4: Confusion matrix: accuracy scores for the (**Adult**) dataset.

Appendix B

Package Dependency

Python version: *3.6.1*

Name Package	***Version***	***Name Package***	***Version***
saliency	0.0.2	anchor	0.0.0.5
tensorflow	1.12.0	deap	1.2.2
numpy	1.16.1	keras	2.2.4
sklearn	0.20.0	lime	0.1.1.32
matplotlib	3.0.0	saliency	0.0.2
pandas	0.23.4	tensorflow-gpu	1.12.0
scikit-image	0.14.0	seaborn	0.9.0
skater	1.0.2		

Table B.1: Package Requirements

Appendix C

Code

```
'''
01 - 10 - 2018
@author: Affolter Riccardo
'''
from datamodel.DataModel import Tab, Img
from algorithms.Properties import Properties
from utility.Constants import Constants
import cv2,time,warnings
import numpy as np

import lime.lime_tabular
from deepexplain.tensorflow import DeepExplain
from LOREM.code.realise import REALISE
from LOREM.code.lorem import LOREM
from LOREM.code.util import neuclidean, multilabel2str
from anchor import anchor_tabular
import saliency
from lime import lime_image
from skater.core.explanations import Interpretation
from skater.model import InMemoryModel

```

```
import tensorflow as tf
from sklearn.model_selection import train_test_split
warnings.filterwarnings("ignore")

class Explainer:

    def __getattr__(self, attr):
            return None

    def addImageRules(self):
        self.rulesImages = []
        self.rulesImages.append([])
        self.rulesImages[0].append([])
        self.rulesImages[0].append([])
        self.rulesImages[0].append([])

    def setRowExplained(self,datamodel,row,existing):
        if not existing:
            if(len(datamodel.features_names) == len(row)):
                self.row = np.array(row) ; return True
            else:
                print("Dimension of the row wrong.")
                return False
        else:
            self.row = datamodel.test_features[row] ; return True

    def setImgExplained(self,img):
        self.addImageRules()
        self.rulesImages[0][0].append(img / 2 + 0.5)
        self.rulesImages[0][1].append("")
        self.rulesImages[0][2].append("Original")

```

```
54     def displayTab(self):
55         self.properties.displayTab()
56 
57     def displayImg(self):
58         self.properties.displayImg()
59 
60     def explain(self,datamodel,num_features,display,methodD,methodS):
61         if(isinstance(datamodel,Tab)):
62             return self.explainTab(datamodel,display)
63         elif(isinstance(datamodel,Img)):
64             return self.explainImg(datamodel,display,methodD,methodS)
65 
66     def newProperties(self):
67         self.properties = Properties()
68 
69     def insertPropertiesTab(self,rules,num_features,time,c,label,
    fidelity,
70         target,class_values):
71         self.properties.Rules = rules
72         self.properties.Length = num_features
73         self.properties.Time = time
74         self.properties.color = c
75         self.properties.label = label
76         self.properties.Fidelity = fidelity
77         self.properties.target = target
78         self.properties.class_values = class_values
79 
80     def insertPropertiesImg(self,img2show,mask,title,time,c,label,exp,
81         shape1, shape2):
82         self.properties.Time = time
83         self.properties.color = c
84         self.properties.label = label
```

```
85          self.properties.exp = exp
86          self.properties.SHAPE1 = shape1
87          self.properties.SHAPE2 = shape2
88          self.rulesImages[0][0].append(img2show)
89          self.rulesImages[0][1].append(mask)
90          self.rulesImages[0][2].append(title)
91          self.properties.rulesImages = self.rulesImages
92
93  class Skater(Explainer):
94
95      def explain(self, model,variables = None ):
96          self.variable_importance(model,variables) if variables != None
97          else self.variable_importance(model)
98
99      def variable_importance(self,model,variables):
100         self.interpreter = Interpretation(model.test_features,
101         feature_names=model.features_names)
102         self.model_global = InMemoryModel(model.cls.predict_proba,
103         examples=model.train_features,target_names=model.class_values)
104         f, ax = self.interpreter.feature_importance.
105         plot_feature_importance(elf.model_global, ascending=False)
106         if(len(model.df.columns) > 15):
107             ax.set_ylim([0, 15])
108         ax.set_xlabel("\u03B5")
109         if variables != -1:
110             self.variable_dependencies(model,variables)
111
112     def variable_dependencies(self,model,x):
113         axes_list = self.interpreter.partial_dependence.
114         plot_partial_dependence([model.features_names[x]],
115         self.model_global,n_samples=100,grid_resolution=30,
116         with_variance=True,figsize=(10, 5))
```

```
117         ax = axes_list[0][1]
118         try:
119             ax.set_title(model.bb_name)
120             ax.set_ylim(0, 1)
121         except AttributeError:
122             ax.suptitle(model.bb_name)
123         ax.set_ylabel("Impact ({})".format(model.features_names[x]))
124 
125 class Lime(Explainer):
126 
127     def explain(self,model,num_features,display,methodD,methodS):
128         if (isinstance(model, Tab)):
129             return self.explainTab(model, display,num_features)
130         elif (isinstance(model, Img)):
131             return self.explainImg(model, display)
132 
133     def display(self, model):
134         if (isinstance(model, Tab)):
135             self.explanation.show_in_notebook(show_table=True,
136             show_all=False)
137             model.toPredict(self.row, self.properties.target)
138             super().displayTab()
139         elif (isinstance(model, Img)):
140             super().displayImg()
141 
142     def calculateFidelity(self,bb_outcome):
143         bb_probs = self.explainer.Zl[:, bb_outcome]
144         lr_probs = self.explainer.lr.predict(self.explainer.Zlr)
145         return 1 - np.sum(np.abs(bb_probs - lr_probs) < 0.01) /
146         len(bb_probs)
147 
148     def explainTab(self,model,display,num_features):
```

```
149         self.newProperties()
150         cls = model.get_cls()
151         if (model.name == Constants.BLACKBOX[2]):
152             cls.probability = True
153         predict_fn = cls.predict_proba
154         t0 = time.clock()
155         self.explainer = lime.lime_tabular.LimeTabularExplainer(model.
156         train_features,class_names=sorted(model.getUniqueValue().
157         tolist()),feature_names=model.features_names,
158         random_state=model.seed,kernel_width=3)
159         self.explanation = self.explainer.explain_instance(self.row,
160         predict_fn, num_features=num_features)
161         t = time.clock() - t0
162         bb_outcome = cls.predict(self.row.reshape(1, -1))[0]
163         bb_outcome_str = model.class_values[bb_outcome]
164         #       label = bb_predict(np.array([X_test[i2e]]))[0]
165         fidelity = self.calculateFidelity(bb_outcome)
166         s = "{ "
167         for n in self.explanation.as_list():
168             s += n[0]
169             s += " , "
170         s = s[:-2] + " }"
171         s +=  " ---> { " + str(bb_outcome_str) + " } "
172         self.insertPropertiesTab(rules=s,num_features=num_features,
173         time = t, c ='r', label ="lime",fidelity = fidelity,
174         target = str(bb_outcome_str),
175         class_values=dict(enumerate(model.class_values)))
176         if(display):
177             self.display(model)
178         return self.properties
179
180     def explainImg(self,img,display):
```

```
        self.newProperties()
        if (self.filename == "mnist" or self.filename == "faces" ):
            image = img.im
            self.explainer = lime_image.LimeImageExplainer()
            t0 = time.clock()
            self.explanation = self.explainer.explain_instance(image,
            img.predict_fn, top_labels=10, hide_color=0,
            num_samples=10000, segmentation_fn=img.segmenter)
            t = time.clock() - t0
            print("Elapsed time (s): ", time.clock() - t0)
            img2show, mask = self.explanation.
            get_image_and_mask(img.x, positive_only=True,
            num_features=5,hide_rest=True)
            self.setImgExplained(image)
            title = 'Pred: {}'.format(img.x)
        else:
            images = img.im; session = img.session; probabilities =
            img.probabilities; processed_images = img.processed_images
            def predict_fn(images):
                return session.run(probabilities,
                    feed_dict={processed_images: images})
            self.explainer = lime_image.LimeImageExplainer()
            t0 = time.clock()
            self.explanation = self.explainer.explain_instance(
            images[0],predict_fn,top_labels=10,hide_color=0,
            num_samples=10000,segmentation_fn=img.segmenter)
            t = time.clock() - t0
            print("Elapsed time (s): ", time.clock() - t0)
            img2show, mask = self.explanation.get_image_and_mask(img.x,
            num_features=5,hide_rest=True)
            self.setImgExplained(img.im[0])
            title = img.names[img.x]
```

```
213         self.insertPropertiesImg(img2show= img2show, mask=mask ,title=
214         title,time=t, c='r' exp="Lime",label="Lime",
215         shape1=img.SHAPE1, shape2=img.SHAPE2)
216         if (display):
217             self.display(img)
218         return self.properties
219
220 class Anchor(Explainer):
221
222     def explainTab(self, model,display):
223         self.newProperties()
224         cls = model.get_cls()
225         categorical = {}
226         t0 = time.clock()
227         self.explainer = anchor_tabular.AnchorTabularExplainer(sorted(
228         model.getUniqueValue().tolist()), model.features_names,
229         model.X, categorical)
230         self.explainer.fit(model.train_features, model.train_labels,
231         model.test_features, model.test_labels)
232         predict_fn = lambda x: cls.predict(x)
233         bb_outcome = predict_fn(self.row.reshape(1, -1))[0]
234         self.explanation = self.explainer.explain_instance(self.row,
235         cls.predict, threshold=0.95)
236         t = time.clock() - t0
237         rullist = self.explanation.exp_map["names"]
238         if(len(rullist) == 0):
239             s = "No rule"
240             fidelity = 0
241         else:
242             s = "{ "
243             for n in rullist:
244                 s += n
```

```
245                 s += " , "
246             s = s[:-2] + " }"
247             s += " ---> { "+ str(model.class_values[bb_outcome]) +" } "
248             fidelity = self.explanation.precision()
249         self.insertPropertiesTab(rules=s,num_features=len(rullist),
250         time=t,c='b',label='Anchor',fidelity=fidelity,
251         target=str(model.class_values[bb_outcome]),
252         class_values=dict(enumerate(model.class_values)))
253         if (display):
254             self.display(model)
255         return self.properties
256
257     def display(self,model):
258         self.explanation.show_in_notebook()
259         model.toPredict(self.row, self.properties.target)
260         super().displayTab()
261
262 class DeepExplainer(Explainer):
263
264     def explainImg(self,img,display,methodD,methodS):
265         self.newProperties()
266         t0 = time.clock()
267         c = False
268         if (self.filename == "mnist" or self.filename == "faces" ):
269             self.properties.Xi =
270             img.xi.reshape(img.im.shape[0],img.im.shape[1])
271             c = True
272         with DeepExplain(session=img.session) as de:
273             if (methodD == "grad*input"):
274                 attributions = {'Gradient * Input':
275                 de.explain('grad*input',img.logits * img.yi if c else
276                 tf.reduce_max(img.logits, 1), img.X, img.xi)}
```

```
277             elif (methodD == 'saliency'):
278                 attributions = { 'Saliency maps':
279                 de.explain('saliency', img.logits * img.yi if c else
280                 tf.reduce_max(img.logits, 1), img.X, img.xi)}
281             elif (methodD == 'intgrad'):
282                 attributions = { 'Integrated Gradients':
283                 de.explain('intgrad', img.logits * img.yi if c else
284                 tf.reduce_max(img.logits, 1), img.X, img.xi)}
285             elif (methodD == 'elrp'):
286                 attributions = { 'Epsilon-LRP':
287                 de.explain('elrp', img.logits * img.yi if c else
288                 tf.reduce_max(img.logits, 1), img.X, img.xi)}
289             elif (methodD == 'deeplift'):
290                 attributions = { 'DeepLIFT (Rescale)':
291                 de.explain('deeplift', img.logits * img.yi if c else
292                 tf.reduce_max(img.logits, 1), img.X, img.xi)}
293             elif (methodD == 'occlusion'):
294                 attributions = { '_Occlusion [1x1]':
295                 de.explain('occlusion',img.logits * img.yi if c else
296                 tf.reduce_max(img.logits, 1),img.X, img.xi)}
297             else:
298                 print("Explainer wrong.")
299                 return
300             print('Done')
301         t = time.clock() - t0
302         self.setImgExplained(img.im)
303         for a,b in enumerate(attributions):
304             explanation = attributions[b][0]
305             title = b
306         self.insertPropertiesImg(img2show=explanation, mask="",
307         title=title, time=t, c='g', exp='DeepExplainer',
308         label=methodD,shape1=img.SHAPE1, shape2=img.SHAPE2)
```

```
309         if (display):
310             self.display(img)
311         return self.properties
312 
313     def display(self,img):
314         super().displayImg()
315 
316 class Saliency(Explainer):
317 
318     def explain(self,img,**kwargs):
319         print("Prediction class: " + str(img.prediction_class))
320         return super().explain(img,**kwargs)
321 
322     def display(self,img):
323         super().displayImg()
324 
325     def explainImg(self,img,display,methodD,methodS):
326         self.newProperties()
327         t0 = time.clock()
328         if (methodS == "Vanilla Gradient"):
329             gradient_saliency = saliency.GradientSaliency(img.graph,
330             img.session, img.y, img.processed_images)
331             vanilla_mask_3d = gradient_saliency.GetMask(img.im,
332             feed_dict={img.neuron_selector: img.prediction_class})
333             smoothgrad_mask_3d=gradient_saliency.GetSmoothedMask(img.im
334             feed_dict={img.neuron_selector:img.prediction_class})
335             explanation =
336             saliency.VisualizeImageGrayscale(vanilla_mask_3d)
337             title = 'Vanilla Gradient'
338             explanation_smoothed =
339             saliency.VisualizeImageGrayscale(smoothgrad_mask_3d)
340             title_smoothed = 'SmoothGrad'
```

```
341         elif (methodS == 'Guided Backprop'):
342             guided_backprop = saliency.GuidedBackprop(img.graph,
343             img.session, img.y, img.processed_images)
344             vanilla_guided_backprop_mask_3d =
345             guided_backprop.GetMask(img.im,
346             feed_dict={img.neuron_selector: img.prediction_class})
347             smoothgrad_guided_backprop_mask_3d=
348             guided_backprop.GetSmoothedMask(img.im,
349             feed_dict={img.neuron_selector: img.prediction_class})
350             explanation= saliency.VisualizeImageGrayscale
351             (smoothgrad_guided_backprop_mask_3d)
352             title = 'Vanilla Guided Backprop'
353             explanation_smoothed =saliency.VisualizeImageGrayscale
354             (vanilla_guided_backprop_mask_3d)
355             title_smoothed = 'SmoothGrad Guided Backprop'
356         elif (methodS == 'Integrated Gradients'):
357             baseline = saliency.np.zeros(img.im.shape)
358             baseline.fill(-1)
359             integrated_gradients = saliency.IntegratedGradients(
360             img.graph, img.session, img.y, img.processed_images)
361             vanilla_integrated_gradients_mask_3d =
362             integrated_gradients.GetMask(img.im,
363             feed_dict={img.neuron_selector:
364             img.prediction_class}, x_steps=25, x_baseline=baseline)
365             smoothgrad_integrated_gradients_mask_3d =
366             integrated_gradients.GetSmoothedMask(
367             feed_dict={img.neuron_selector: img.prediction_class},
368             x_steps=25, x_baseline=baseline)
369             explanation =saliency.VisualizeImageGrayscale
370             (vanilla_integrated_gradients_mask_3d)
371             title = 'Vanilla Integrated Gradients'
372             explanation_smoothed = saliency.VisualizeImageGrayscale(
```

```
373             smoothgrad_integrated_gradients_mask_3d)
374             title_smoothed = 'Smoothgrad Integrated Gradients'
375         elif (methodS == 'Occlusion'):
376             occlusion_gradients = saliency.Occlusion(img.graph,
377             img.session, img.y, img.processed_images)
378             vanilla_occlusion_gradients_mask_3d=
379             occlusion_gradients.GetMask(img.im,
380             feed_dict={img.neuron_selector:img.prediction_class})
381             smoothgrad_occlusion_gradients_mask_3d =
382             occlusion_gradients.GetSmoothedMask(img.im,
383             feed_dict={img.neuron_selector: img.prediction_class})
384             explanation =saliency.VisualizeImageGrayscale
385             (vanilla_occlusion_gradients_mask_3d)
386             title = 'Vanilla Occlusion'
387             explanation_smoothed = saliency.VisualizeImageGrayscale
388             (smoothgrad_occlusion_gradients_mask_3d)
389             title_smoothed = 'Smoothgrad Occlusion'
390         else:
391             print("Explainer wrong.")
392             return
393         t = time.clock() - t0
394         self.setImgExplained(img.im)
395         self.insertPropertiesImg(img2show=explanation, mask="",
396         title=title, time=t, c='g', exp='Saliency',
397         label=methodS,shape1=img.SHAPE1, shape2=img.SHAPE2)
398         self.rulesImages[0][0].append(explanation_smoothed)
399         self.rulesImages[0][1].append("")
400         self.rulesImages[0][2].append(title_smoothed)
401         if (display):
402             self.display(img)
403         return self.properties
```

Bibliography

[1] M. Ancona, E. Ceolini, C. Oztireli, and M. Gross. Towards better understanding of gradient-based attribution methods for deep neural networks. In *International Conference on Learning Representations*, 2018.

[2] E. Angelino, N. Larus-Stone, D. Alabi, M. Seltzer, and C. Rudin. Learning certifiably optimal rule lists. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 35–44, New York, NY, USA, 2017. ACM.

[3] L. Arras, F. Horn, G. Montavon, K. Müller, and W. Samek. Explaining predictions of non-linear classifiers in NLP. *CoRR*, abs/1606.07298, 2016.

[4] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Muller, and W. Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLOS ONE*, 10(7):1–46, 2015.

[5] T. Brennan, W. Dieterich, and B. Ehret. Evaluating the predictive validity of the compas risk and needs assessment system. *Criminal Justice and Behavior*, 36(1):21–40, 2009.

[6] J. Burrell. How the machine 'thinks': Understanding opacity in machine learning algorithms. *Big Data & Society*, 3(1):2053951715622512, 2016.

[7] P. Choudhary, A. Kramer, and datascience.com team. datascienceinc/Skater: Enable Interpretability via Rule Extraction(BRL), Mar. 2018.

[8] C. R. de Sá, W. Duivesteijn, P. J. Azevedo, A. M. Jorge, C. Soares, and A. J. Knobbe. Discovering a taste for the unusual: exceptional models for preference mining. *Machine Learning*, 107(11):1775–1807, 2018.

[9] R. C. Deo. Machine learning in medicine. *Circulation*, 132(20):1920–1930, 2015.

[10] J. Dodge, Q. V. Liao, Y. Zhang, R. K. E. Bellamy, and C. Dugan. Explaining models: An empirical study of how explanations impact fairness judgment. In *Proceedings of the 24th International Conference on Intelligent User Interfaces*, IUI '19, pages 275–285, New York, NY, USA, 2019. ACM.

[11] F. Doshi-Velez and B. Kim. A roadmap for a rigorous science of interpretability. *CoRR*, abs/1702.08608, 2017.

[12] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118, 2017.

[13] A. A. Freitas. Comprehensible classification models - a position paper. *ACM SIGKDD Explorations*, 15(1):1–10, 2013.

[14] R. Guidotti, A. Monreale, S. Ruggieri, D. Pedreschi, F. Turini, and F. Giannotti. Local rule-based explanations of black box decision systems. *arXiv preprint arXiv:1805.10820*, 2018.

[15] S. Hara, K. Ikeno, T. Soma, and T. Maehara. Maximally invariant data perturbation as explanation. *arXiv preprint arXiv:1806.07004*, 2018.

[16] M. Hardt, E. Price, and N. Srebro. Equality of opportunity in supervised learning. *CoRR*, abs/1610.02413, 2016.

[17] U. Johansson, C. Sönströd, U. Norinder, and H. Boström. Trade-off between accuracy and interpretability for predictive in silico modeling. *Future medicinal chemistry*, 3:647–63, 2011.

[18] I. KONONENKO. Inductive and bayesian learning in medical diagnosis. *Applied Artificial Intelligence*, 7(4):317–337, 1993.

[19] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[20] T. Lombrozo. The structure and function of explanations. *Trends in cognitive sciences*, 10(10):464–470, 2006.

[21] A. Mahendran and A. Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. *Int. J. Comput. Vision*, 120(3), 2016.

[22] B. Mittelstadt, C. Russell, and S. Wachter. Explaining explanations in ai. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, FAT* '19, pages 279–288, New York, NY, USA, 2019. ACM.

[23] C. Molnar. Interpretable machine learning. *A Guide for Making Black Box Models Explainable*, 2018.

[24] M. Narayanan, E. Chen, J. He, B. Kim, S. Gershman, and F. Doshi-Velez. How do humans understand explanations from machine learning systems? an evaluation of the human-interpretability of explanation. *CoRR*, abs/1802.00682, 2018.

[25] J. Nayak, B. Naik, and H. Behera. A comprehensive survey on support vector machine in data mining tasks: applications & challenges. *International Journal of Database Theory and Application*, 8(1):169–186, 2015.

[26] 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*, pages 1135–1144. ACM, 2016.

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

[28] S. R. D. P. F. T. Riccardo Guidotti, Anna Monreale and F. Giannotti. A survey of methods for explaining black box models. *ACM Computing Surveys (CSUR)*, 2018.

[29] M. Robnik-Šikonja and M. Bohanec. Perturbation-based explanations of prediction models. In *Human and Machine Learning*, pages 159–175. Springer, 2018.

[30] A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje. Not just a black box: Learning important features through propagating activation differences. *CoRR*, abs/1605.01713, 2016.

[31] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[32] Q. Sun and G. DeJong. Explanation-augmented svm: An approach to incorporating domain knowledge into svm learning. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 864–871, New York, NY, USA, 2005. ACM.

[33] M. Sundararajan, A. Taly, and Q. Yan. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3319–3328. JMLR. org, 2017.

[34] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[35] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[36] S. B. Thacker. Meta-analysis: a quantitative approach to research integration. *JAMA*, 259(11):1685–1689, 1988.

[37] M. van Lent, W. Fisher, and M. Mancuso. An explainable artificial intelligence system for small-unit tactical behavior. In *AAAI*, 2004.

[38] S. Wachter, B. D. Mittelstadt, and C. Russell. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *CoRR*, abs/1711.00399, 2017.

[39] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[40] Q. Zhang, Y. N. Wu, and S.-C. Zhu. Interpretable convolutional neural networks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8827–8836, 2018.

One sees clearly only with the heart. What is essential is invisible to the eye.

Le Petit Prince