Alma Mater Studiorum - Università di Bologna

DOTTORATO DI RICERCA IN

DATA SCIENCE AND COMPUTATION

Ciclo 34

Settore Concorsuale: 09/H1 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

Settore Scientifico Disciplinare: ING-INF/05 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

MEANINGFUL INSIGHTS: EXPLAINABILITY TECHNIQUES FOR BLACK-BOX MODELS ON TABULAR DATA

Presentata da: Giorgio Visani

Coordinatore Dottorato

Daniele Bonacorsi

Supervisore

Federico Chesani

Co-supervisore

Marco Roccetti

Esame finale anno 2023

Alma Mater Studiorum - University of Bologna
Department of Computer Science

Ph.D. Thesis
January 31st, 2023

Meaningful Insights: Explainability Techniques for Black-box Models on Tabular Data

Giorgio Visani

Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy (Computer Sciences)

Advisors:

Prof. Federico Chesani
Department of Computer Science, University of Bologna

Dr. Enrico Bagli
Crif S.p.A.

Abstract

Artificial Intelligence (AI) and Machine Learning (ML) are novel data analysis techniques providing very accurate prediction results. They are widely adopted in a variety of industries to improve efficiency and decision-making, but they are also being used to develop intelligent systems. Their success grounds upon complex and non-linear mathematical models, whose decisions and rationale are usually difficult to comprehend for human users to the point of being dubbed as black-boxes. This is particularly relevant in sensitive and highly regulated domains.
To mitigate and possibly solve this issue, the Explainable AI (XAI) field became prominent in recent years. XAI consists of models and techniques to enable understanding of the intricated patterns discovered by black-box models.

In this thesis, we consider model-agnostic XAI techniques, which can be applied to Tabular data, with a particular focus on the Credit Scoring domain. Special attention is dedicated to the LIME framework, for which we propose several modifications to the vanilla algorithm, in particular: a pair of complementary ***Stability Indices*** that accurately measure LIME stability, and the ***OptiLIME*** policy which helps the practitioner to find the proper balance among the explanations' stability and reliability. We subsequently put forward ***GLEAMS*** a model-agnostic surrogate interpretable model which requires to be trained only once, while providing both Local and Global explanations of the black-box model. It is capable of producing feature attributions and what-if scenarios, from both a dataset and model perspective. Eventually, we argue that synthetic data are an emerging trend in AI, being more and more used to train complex models instead of original data. To be able to explain the outcomes of such models, we must guarantee that synthetic data are reliable enough to be able to translate their explanations to real-world individuals. To this end we propose ***DAISYnt***, a suite of tests to measure synthetic tabular data quality and privacy.

Acknowledgments

To Federico and Enrico, which constantly supported, encouraged and cheered me up during the ups and downs of this journey, I couldn't ask for better guidance. To Clara and Gianluca which supervised me abroad, making me feel welcome and reminding me of our roots. To Sara, which helped me out with my burocracy struggles.

To the big University family with which we shared so many things. Fabri and Fede, I had you by my side from the beginning, Matti and Fede joined the team one year later but proved to be solid additions, the old top blokes Anto, Andrea and Andrea always ready to share tips during the day as well as spirits at night. To Giacomo and Robert, sharing the office together and putting up with me even when I proved to be stubborn. To Adrian, Elias and Gio my Bruxelles' buddys, as well as to the Sydney mates that I'm getting to know at the time of writing.

To my addictive Bologna's friends, thanks for your classy and finest sense of humor which makes me eagerly wait for our weekly reunion.

To my good ol' friends, everytime I'm heading back home our shared memories fill my mind. High school times shine as if it was yesterday, no more than our recent European Parliament expedition to stand out and make our (loud) voices heard.

To my family that always supported me with love and sweetness, Mom and Dad I hope I'll live up to be a brilliant parent as you, when my time comes. Marco and Alice, they say big brothers should lead by example but you gave me more love than I did. I'm lucky to have you as siblings and I'm proud of your milestones as if they were mine, keep up the good work!

It takes patience to read through it all, but even more patience to keep it up whenever I'm far away. *Dulcis in fundo*, Vale your are indeed the sweetest and most important person in my life. You always standed by my side as my best friend, my confidant, my new family. I won't let you down Meow.

Contents

Chapter 1

Introduction

Being able to predict the future has long been regarded as the "holy grail" of human society. The idea of using past experience as a key to inferring future behaviour dates back an immemorable time. Among others to recognize that, Confucius over 2500 years ago stated "Study the past if you would define the future".
Nonetheless, only in relatively recent times this concept has been formalized mathematically into the prediction models field. First examples date back to the end of the 19th century, when Sir Francis Galton and subsequently Karl Pearson, developed Linear Regression for the first time. Since then, a variety of mathematical structures were put forward to model different phenomena. At the same rate, better computational tools allowed for more demanding calculations, up to the present date in which computational power at disposal is believed to increase exponentially over the years.

These factors created fertile ground for the Machine Learning (ML) field, which includes algorithms analyzing patterns and structures in data to enable learning, reasoning, and decision-making without human interaction required. ML became soon pervasive and widespread across multiple domains, making it one of the pillars of the present interconnected society. At the same time, the complexity of the ML prediction models constantly increased, until the trend was widely recognized and the term "black-box" model, i.e. prediction model with highly complicated model internals, was coined. The programmatic steps of a black-box model are usually known, but it is difficult to grasp how the mathematical function obtains certain prediction values. Such peculiarity is due to extremely complicated functions employed, iterative algorithms continuously refining the function and preventing human beings from staying on par with the modifications, or a combination of the two.

Legal Requirements

However, the new paradigm needs necessarily to leverage sensitive information, as required by novel regulations. In the European Union, the General Data Protection Regulation (GDPR) [57] was enforced in 2018, introducing the concept of the "Right to an Explanation", i.e. each individual affected by an Algorithm's decisions has the right to know the model's rationale in predicting the specific pattern. In 2019 the High-Level Expert Group

on Artificial Intelligence (AI) presented the "Ethical Guidelines for trustworthy AI" [55], containing the three pillars of AI trustworthiness, i.e. being lawful, ethical and robust, along with 7 key requirements that AI systems should meet. Eventually, in 2021 the European Parliament drafted the first proposal for the Artificial Intelligence Act (AIA) [26], regarded as the first law on AI by a regulator anywhere. Even if the "Old World" has been very active in the regulatory aspect of AI, other major countries put forward similar requirements. With no intention of being exhaustive, we remind only the U.S. FERPA [33] and HIPAA [17], educational and medical data regulations respectively.

Even if quite compelling, compliance with existing regulations is only one of the drivers behind ***Explainability***. In fact, ***Interpretability*** can help with other relevant ML issues, such as: i) the ability to debug unexpected behaviours of the models, as well as ii) to grasp the true relationship among the collected data. The former bears importance from a practitioner's point of view, for understanding how one can improve the model. The latter is of particular interest in the hard sciences, where mathematical models are employed to shape a given physical, medical or financial phenomenon. Insights on the true Data Generating Process (DGP) are hence essential. All such tasks cannot be easily carried out, relying just on powerful black-box prediction models.

Crif

The effort produced in this thesis represents 4 years of research carried out in joint collaboration between the University of Bologna and Crif S.p.A. -*from now on just Crif*-, a global company headquartered in Bologna. Crif's core business is Credit Scoring, namely the evaluation of the probability that a debtor will not repay the due amount. However, Crif is specialized in business information, credit solutions and outsourcing and processing services as well. In fact, Crif is fully committed to providing help and support to any legal financial institution, by means of advanced data analysis.
Its expertise in Credit Scoring dates back to the 80s, making the company one of the leaders in the Italian credit bureau market as well as an important benchmark worldwide. Nowadays, one of Crif's endeavours is the adoption of advanced analytics for Credit Scoring.

Statistical approaches have been successfully exploited in Credit Risk since long, becoming the gold standard. However recently, also machine and deep learning techniques have been applied to the task, showing an important increase in prediction quality and performance. Albeit the improved ML performances, Credit Scoring is a highly regulated domain, in which a single decision can have significant impacts on people's lives. The ***European Banking Authority*** (EBA) redacted a specific Report [23] on best practices to use Artificial Intelligence in Credit Scoring. Also the Artificial Intelligence Act takes a stance on Credit Scoring, classifying it as an AI high-risk application. This still allows for AI to be used in this business, but the process should comply with a copious list of requirements and best practices, prominent among them is the ability to provide explanations for the decisions taken.

Tabular Data

The vast majority of Banking data is in Tabular form and relatively low-dimensional, i.e. they usually contain a few dozen features while some datasets may include up to 100 or 200 variables.

Tabular Data are data which can be represented in a table: each row stands for a specific observation, while each column contains information on a given variable. We will refer to the generic i-th unit -*row*- of the dataset as $x^{(i)}$, while the j-th column of the same generic unit will be addressed as $x_j^{(i)}$. The j-th column vector containing values of a specific variable is labelled X_j, while the entire dataset is addressed as X. In Tabular Data the variables are usually meaningful on their own: the variables ***Income***, ***Loan Intent*** or ***Own House*** in Table 1.1 represent specific traits of the individuals and are quite informative for an end user. This property enables explanations based on the input features directly. However, considering different data sources, it is not always true that original variables are meaningful. As an example, an Image is usually stored as a 3-dimensional tensor, where each entry corresponds to the intensity of the colour of a single pixel. In this case, the information of the intensity colour of a specific pixel is generally difficult to relate to any insight about the image itself -*except for trivial cases*-.

Age	Income	Own House	Loan Intent	Loan Amount	Loan/Income Ratio	Default
22	59000	Rent	Personal	35000	0.59	Yes
21	9600	Own	Education	1000	0.10	No
25	9600	Mortgage	Medical	5500	0.57	No
23	65500	Rent	Medical	35000	0.53	No

Table 1.1: Toy Credit Scoring Dataset, described in detail in Chapter 3.4.1

Information contained in tabular variables can be of two different types: categorical or numerical. Referring to Table 1.1, the variable ***Income*** is numerical, i.e. it assumes values in a subset of the real line $\mathbb{R}$. Examples of categorical variables are instead ***Loan Intent*** and ***Default***, whose domain is represented by a set of different choices or categories -*which in turn might be represented as integer numbers*-. In particular, in the case of ***Default***, the categories are only 2 and the variable is also called dummy, flag or binary variable. The first step of any prediction model is to transform categorical variables into numerical ones. This step is usually not incorporated in the Machine Learning model itself -*apart for few recent models, like the Transformers language models [111]*-, but the choice on the categorical encoding algorithm to use is left to the practitioner.
Such a compact representation in table form allows for simple storage of Tabular datasets.

Contributions

The scope of this thesis is restricted to ***post-hoc*** Explainability techniques, applied to Tabular Data.

By post-hoc, we mean explanations that can be applied separately from the training procedure of the models -*explained in greater detail in Chapter 4*-. In particular, they can be used on already trained models, making ***post-hoc*** explanations especially suitable in business processes where a prediction model has been already deployed in a production environment. With the same goal, we will mostly consider only ***model-agnostic*** frameworks, namely techniques able to explain any kind of prediction model. Such techniques may enable out-of-the-box explainability and possibly law compliance for any company's prediction model. Regarding the restriction to the Tabular data domain, such choice has been made to readily employ the techniques on Credit Scoring data. Nonetheless, all the work presented here is not limited to a specific business field, on the contrary it is general and valid for any database respecting the above-mentioned requirements.

The main contributions of this thesis entail:

A thorough and in-depth analysis of the vanilla LIME explainability framework [93], in particular showing its application to the Credit Scoring domain and highlighting its caveats and specific situations in which LIME is bound to fail. We focus on the stability of the explanations, i.e. namely repeated applications of the method under the same conditions may obtain very different results. In order to help the practitioner to spot instability, we devise a pair of complementary stability indices tailored specifically on LIME. We also tackle the problem of jointly optimizing the stability and the adherence -*fidelity of the explanations towards the black-box model*- of LIME explanations. The result is the OptiLIME policy, which automatically selects the best kernel width -*main LIME hyper-parameter*-, to achieve the goal put forward. This line of work specifically related to the LIME technique is described in:

LIME applied to Credit Scoring Giorgio Visani, Federico Chesani, Enrico Bagli, Davide Capuzzo and Alessandro Poluzzi. *Explanations of Machine Learning predictions: a mandatory step for its application to Operational Processes*, 16^{th} Credit Scoring and Credit Control Conference (CRC) August 28-30, 2019, Edinburgh, UK

LIME Stability Indices Giorgio Visani, Enrico Bagli, Federico Chesani, Alessandro Poluzzi and Davide Capuzzo. *Statistical stability indices for LIME: obtaining reliable explanations for Machine Learning models*, Journal of the Operational Research Society, Vol. 73.1 Pag. 91-101, 2022. Taylor & Francis Publishing.

OptiLIME Giorgio Visani, Enrico Bagli, Federico Chesani. *Optilime: Optimized lime explanations for diagnostic computer algorithms*, Proceedings of the CIKM 2020 Workshops, October 19-20, 2020, Galway, Ireland. Vol 2699

Subsequently, we introduce GLEAMS (Global and Local ExplAnations through Model Space partitioning), which aims at bridging the gap between local and global explanation frameworks -*the two frameworks are described in Chapter 4*-. GLEAMS retains the best of

both worlds, allowing for precise and accurate local explanations while also giving insights about the entire variable space. GLEAMS is described in

GLEAMS Giorgio Visani, Damien Garreau, Vincenzo Maria Stanzione. *GLEAMS: Bridging the Gap Between Local and Global Explanations*, 2022, currently under review.

Eventually, we argue that Synthetic Data is a prominent trend in recent Machine Learning and more and more analytic companies train complex models on such generated instances. To allow explainability in this setup, we need to guarantee the Synthetic Data quality from different viewpoints. For this reason, we put forward a taxonomy of the relevant concepts, along with an array of tests to quantify such concepts on a given synthetic dataset. The tests are collected in DAISYnt, a dedicated test suite disclosed in

DAISYnt Giorgio Visani, Giacomo Graffi, Mattia Alfero, Enrico Bagli, Davide Capuzzo and, Federico Chesani. *Enabling Synthetic Data adoption in regulated domains*, Proceedings of the DSAA 2022 Application Track, October 13-16, 2022, Online.

The rest of the thesis is as follows: in Chapter 2 we introduce prediction models, distinguishing between Statistical and Machine Learning models. In Chapter 4 we provide an extensive review of the literature on ***post-hoc***,***model-agnostic*** explanations; in Chapter 5 we thoroughly describe the LIME technique and its weakpoints; in Chapter 6 we introduce a pair of stability indices specifically tailored on LIME; while in Chapter 7 we discuss how to fine-tune LIME main hyper-parameter using the OptiLIME policy. Chapter 8 is dedicated to the GLEAMS method, combining local and global explanations in a unified framework; in Chapter 9 we put forward DAISYnt, a suite of tests to assess the quality of synthetic data. Chapter 10 conclude the dissertation, with a brief summary and discussion.

Chapter 2

Prediction Models

Since the ***Interpretability*** problem is inherent to the complex prediction models, we will start the journey into ***Explainability*** by understanding how some prediction models are considered explainable by default, while others suffer from lack of clarity -*to the point of being called black-box*-. In particular, we consider the Parametric Statistical Models as opposed to the class of Machine Learning models. We argue that constraints on the learnt geometrical surface help the Parametric models achieve explainability through relatively simple mathematical formulas. Machine Learning models, instead, usually produce quite flexible but not easily understandable surfaces, due to too complex or even not specified mathematical formulations of the surface. A notable exception is the Decision Tree Model, which even if non-parametric and flexible, benefits from the fact that iterative splits can be seen both as a piecewise constant function in the geometrical space or as a set of rules. The latter provides a way of interpreting the model behaviour, as already well-explored into the symbolic and logic community inside the Artificial Intelligence field.

The Chapter is structured as follows: in 2.1 a general introduction on the Prediction Models and how it is always possible to represent a model on Tabular Data as a surface in the geometrical space of the variables, Chapter 2.2 concerns with Statistical Parametric Models, considered explainable by default, Chapter 2.3 contains some of the most well-known Non-Parametric Machine Learning models

2.1.0. Prediction Models in general

Prediction models are mathematical functions defining the relationship between values of some auxiliary variables X, also called regressors or independent variables, and the values of a target variable Y, i.e. the event or quantity of interest. Depending on the Y domain, a continuous target variable yields Regression models, while a categorical one requires Classification models -*Binary or Multi-class Classification models, based on the number of Y categories*-.

More formally, a prediction model is a multivariate function $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X} \in \mathbb{R}^{d+1}, \mathcal{Y} \in \mathbb{R}$ are the subspaces where the X, Y random variables respectively live, d is the number of independent variables. Interestingly, the $\mathcal{X}$ subspace has dimension $d + 1$ to

allow f to model a constant term different from 0.
In statistics, data are assumed to be generated from a Data Generating Process (DGP) combined with a source of white noise, so that the standard formulation of the problem is $Y = DGP(X) + \mathcal{E}$, where $\mathcal{E} \sim N(0, \sigma^2)$. Any prediction model f strives to approximate the DGP, i.e. the underlying unknown distribution which generated the data. In particular, standard models approximate the conditional expectation of the target variable $\mathbb{E}[Y|X]$ in Regression settings -*continuous* Y- or the conditional probability of a specific class $\mathbb{P}[Y = j|X]$ in Classification -*categorical* Y-.

Most of the models refine their predictions using optimization algorithms and this may lead to overfitting, i.e. to learn specific patterns of the given dataset, which are not a general characteristic of the DGP. To avoid overfitting, it is usual practice to divide the data into train and test set, respectively called D_{train}, D_{test} from now on. The model is trained on the D_{train} dataset, but the final performance results are displayed on D_{test}. A high discrepancy among performance on D_{train} and D_{test} is a clear sign of overfitting.
Moreover, complex models usually have hyper-parameters governing the training phase. These parameters require tuning, which may again be a source of overfitting. For this reason, such models require an additional splitting to produce the validation dataset D_{val} which is only used to choose the best hyper-parameter set.

2.2.0. Classical Statistical Models

The classical methodology is based on a variety of techniques stemming from the statistics field, the most popular ones are Linear, Logistic and Probit models.

Linear Regression

Linear Regression generates the f function as a linear hyperplane:

$$\mathbb{E}[Y|X] = X^T\beta$$

where β is the coefficient vector parametrizing each X variable.

Linear Regression is generally used in Regression settings, where $\mathcal{Y} = \mathbb{R}$ -*unrestricted* f *codomain*-. It might be employed even in classification settings, but this time $\mathcal{Y}$ represents a conditional probability in $[0, 1]$ while Linear Regression has no restriction on the outputs, i.e. it can produce values outside the required interval. A simple way to solve this issue is to clip extreme values to the closest valid probability value.
Linear Regression's main strength is the ease of estimation and explanation, although the linearity is a too strict assumption in many cases.

Logistic and Probit Regression

On the other hand, Logistic and Probit models are mainly used for Classification purposes, thanks to the transformation applied to Y to transform it into a new variable spanning over the entire real line $\mathbb{R}$. -*In the more general setting of Generalized Linear Models (GLM) this transformation is called link function*-.

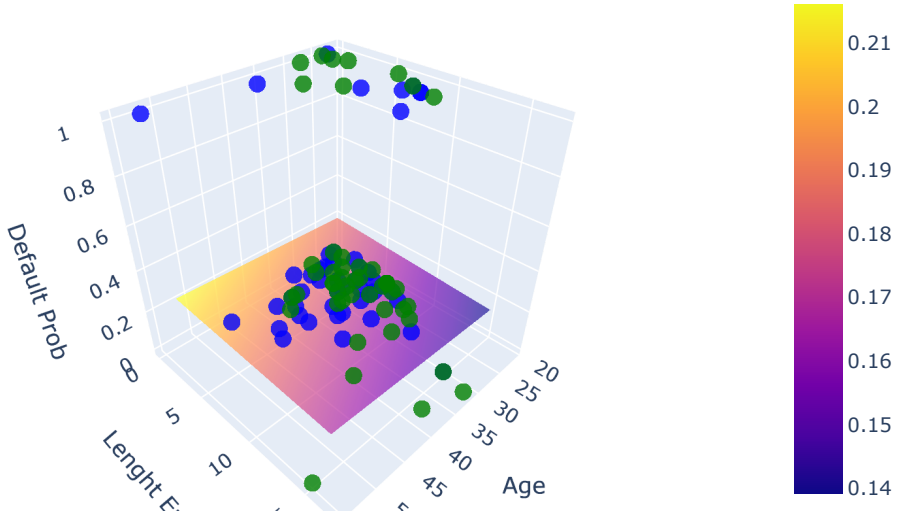


Figure 2.1: Linear Regression Surface obtained on the Toy Credit Scoring Dataset (Chapter 3.4.1). The model is trained to predict the ***Default Probability*** using ***Age*** and ***Lenght of Employment***

The transformation is a bijective function, meaning that it is always possible to convert each value of the new variable back into the probability value that generated it.

$$\Pr(Y=1|X) = \begin{cases} \dfrac{\exp(X^T\beta^{(1)})}{1+\exp(X^T\beta^{(1)})} & \text{Logistic Model} \quad (2.1) \\ \Phi(X^T\beta^{(2)}) & \text{Probit Model} \quad (2.2) \end{cases}$$

$\beta^{(1)}, \beta^{(2)}$ are the parameters respectively of the Logistic and Probit Models.
$\Phi(\cdot)$ is the Cumulative Distribution Function of a standard Gaussian $\mathcal{N}(0,1)$. In this formulation, we highlight the inverse transformation, mapping the $X^T\beta$ matrix into the $[0,1]$ interval.

Both Logistic and Probit have the additional advantage of modelling the relation in a non-linear way. This is a dramatic increase in representation capability, even if such curvy surfaces are bound to be monotonically increasing or decreasing (as shown in Figure 2.2).

An additional perk of Logistic Regression, when compared to Probit, is its interpretability of results: the parameters derived from the best curve's estimation, can be regarded

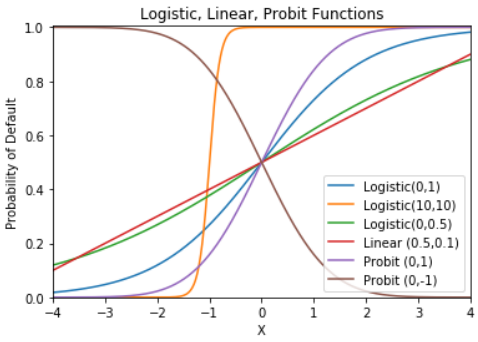


Figure 2.2: Shape of Logistic, Probit and Linear functions, associated with different parametrization. In this easy setup, the Probability of Default (PD) is modelled against a single independent variable X.

as odds ratio, i.e. the ratio between the probability of default and non-default, namely $\frac{P(Y=1|X=x)}{P(Y=0|X=x)}$.

Starting from the mean value of one specific independent variable, the increase of 1 unit brings an increase in the odds ratio that is equal to the exponentiated parameter. The relation is valid when the other regressors' values are fixed to their mean.
This benefit is due to the particular transformation employed by the model, which preserves the chance of interpreting the results.

2.3.0. Machine Learning

The classical definition of Machine Learning dates back to 1997 on behalf of Tom Mitchell [81]:

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.

By this train of thought, almost any kind of prediction algorithm may fall into the class of Machine Learning models. Consider Logistic Regression, the parameter tuning phase is done through an iterative algorithm, usually Newton-Raphson, which improves the estimated model's performance at each iteration, measured by the increase in the Likelihood value.

Because of the extent of such a general framework, in this dissertation we consider only non-parametric Machine Learning models. They estimate the relationship between the target variable and the predictor variables, without constraining the surface to have a precise functional form. This peculiarity allows to model non-linear relations of any possible shape, making the technique more flexible compared to classical parametric methods. Machine Learning models of this kind usually outperform classical methods in non-linear settings and achieve the same results when the nature of true relations is simply linear.

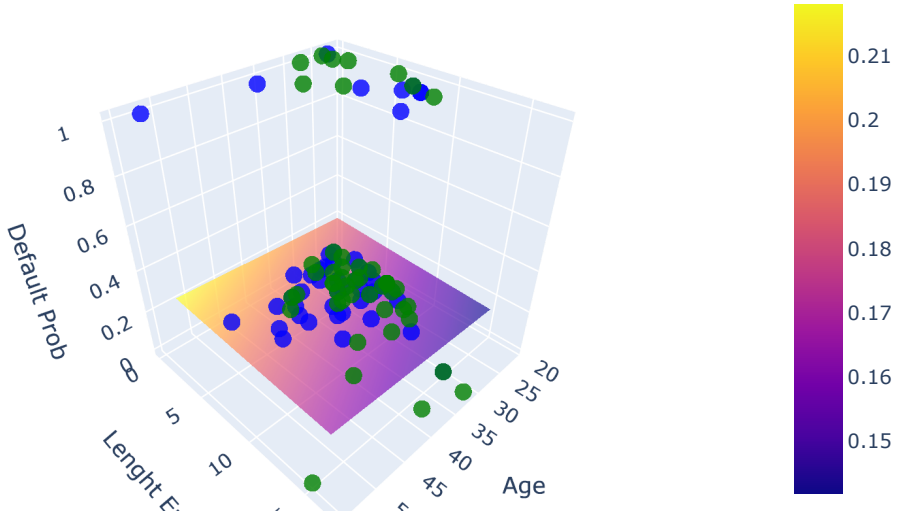


Figure 2.3: Logistic Regression Surface obtained on the Toy Credit Scoring Dataset (Chapter 3.4.1). The Logistic Function behaves very similar to Linear Regression -*monotonicity is probably too strict assumption in this case*-

Hereafter, a brief and non-exhaustive list of some of the most well-known Machine Learning models:

Neural Networks

The neural network is a Machine Learning model, inspired by the human brain structure. The model's building block is the Perceptron [95], which consists of a mathematical function taking a vector of inputs $x^{(i)}$, i.e. a specific D_{train} unit's values, computing their weighted sum and transforming them through an activation function *act* to obtain a single output $y^{(i)}$:

$$y^{(i)} = act\left(\sum_j w_j x_j^{(i)}\right)$$

where w_j is the weight assigned to the j-th element of the input vector, while *act* is the chosen activation function.
The Neural Network (NN) model exploits a high number of perceptrons to create an interconnected network (Figure 2.4), typically structured in an input layer -*the original dataset variables* X_j-, one or more hidden layers, and an output layer -*the final prediction value*-. Hidden layers are similar to the input layer, but the input to each neuron is the

output of the previous layer.

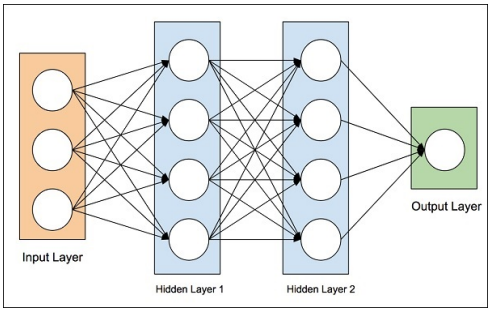


Figure 2.4: Basic Structure of a Multi Layer Perceptron (MLP) Neural Network

There exist many different Neural Network models, mainly differing in their architecture, depending on which layers are employed and how they are combined to achieve the prediction. State-of-the-art NN are Multi Layer Perceptrons (MLP), Convolutional Neural Networks (CNN), Long Short Term Memory (LSTM) and many others. However, we are going to focus on MLP models which are readily applicable to tabular data domains.
The activation function plays an important role in the NN framework, being responsible for modelling the non-linearity of the resulting prediction surface. Many different choices are available, with ReLU, Sigmoid, and Tanh being the best well-known ones.
In particular, the ***Sigmoid***, or Logistic function, is defined as $act(x) = \frac{1}{1+e^{-x}}$ and allows to model a monotonic non-linear function in the interval $[0, 1]$. The ***ReLU*** (Rectified Linear Unit) function is defined as $act(x) = \max(0, x)$ and produces a piecewise linear curve. Eventually, the ***Tanh*** (hyperbolic tangent) has formula $act(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ and produces a curve similar to the sigmoid function, but mapping input values to the range $[-1, 1]$. Each neuron contributes to the non-linearity of the network, by combining the relatively simple functions in a complex highly non-linear surface. In particular, using Sigmoid or Tanh activations we achieve smooth surfaces (Figure 2.5), while ReLU yields discontinuous surfaces. The Neural Network framework has the potential to perfectly approximate any unknown mathematical function, under mild conditions [59].

The most common algorithm to train a neural network is called ***backpropagation***, which adjusts the weights w_j of the connections between neurons. Backpropagation consists in computing the NN prediction error for each $x^{(i)}$ unit and differentiating it with respect to the network's weights of each layer. This is done using the Chain rule, which allows us to propagate the error's gradients to the previous layers. The weights are updated according to the computed gradients, iteratively, until a stopping criterion is met. Given the significant approximation capabilities, Neural Networks are prone to overfitting. Therefore, a validation set is useful to stop the procedure before learning too specific patterns.

The huge number of nonlinear perceptrons combined together yields very complex models, which possibly count millions of parameters. The mathematical formulation of the surface is hence too complex to be considered interpretable.

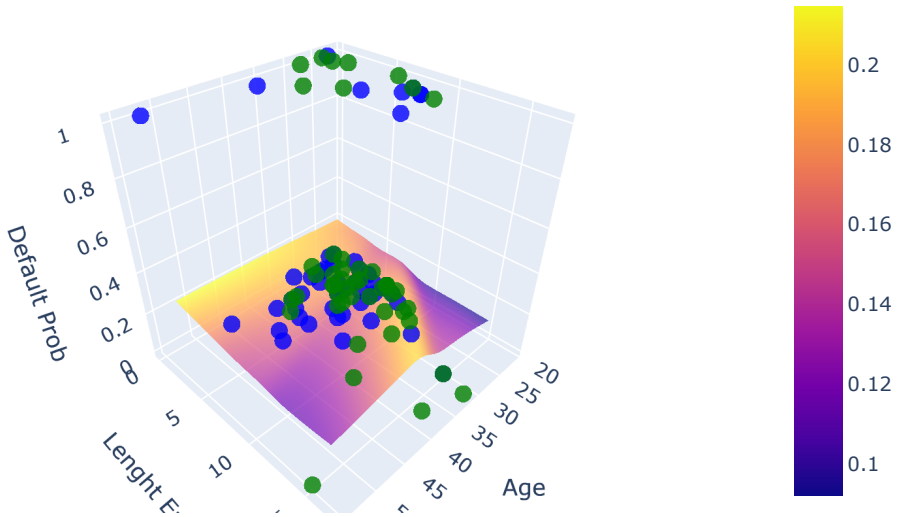


Figure 2.5: Neural Network Surface obtained by an MLP model equipped with Sigmoid activation function. The model has been trained on the Toy Credit Scoring Dataset (Chapter 3.4.1)

Decision Trees

Decision Trees are generally considered one of the simplest Machine Learning models, and interpretable models themselves.
The decision function is iteratively built, by choosing a single variable X_j and a split point x_j , which yields two mutually exclusive and complementary partitions of $\mathcal{X}$. The best split is chosen by inspecting any possible split-point on any independent variable and comparing the related Splitting Criterion values. There exist many different choices for the splitting criterion, among them Variance Reduction is useful for regression problems; while Information Gain, Gini or the Chi-Square p-value are valid choices when it comes to classification [49]. From a geometric point of view, the Tree divides $\mathcal{X}$ in hyper-rectangles by means of splits parallel to the coordinate axes -*independent variables*-. In the two generated partitions, the process is repeated until a stopping criterion is met. On the final partitions $R_1, \ldots, R_k$, the Decision Tree computes the average Y value and assigns it to each unit in the specific region.

The f model surface is a piecewise constant function, with discontinuities in correspondence with each split-point that prevent the existence of the gradient. Although the structure of the Tree itself can be seen as a set of decision rules, in fact each path con-

necting the root node with a specific leaf is fully described by each intermediate split and associated rule -*whether the unit value is higher or lower than* x_j -. The Tree predicts a new unit $x^{(i)}$ by determining its corresponding partition R_i using the appropriate rule. This rule is regarded as the explanation for the unit $x^{(i)}$ since it is clear and human-understandable, namely it consists of a set of conditions the unit $x^{(i)}$ must abide to get that specific prediction.

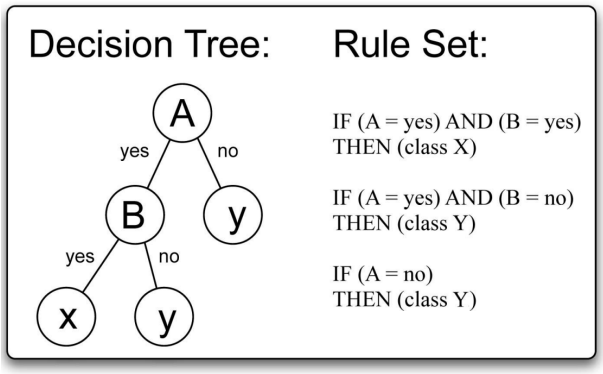


Figure 2.6: The Decision Tree model can be seen as consecutive splits starting from the root node to the leaves (left), or as a set of decision rules (right).
Courtesy of [27]

Decision Trees can obtain high accuracy at the cost of pronounced overfitting when the Tree is very deep. On the contrary, a shallow Tree is usually not overfitted but also not as precise in prediction. To leverage this trade-off, ensemble models -*aggregations of several models together*-, based on vanilla Decision Trees have been proposed. In particular, Random Forest and Gradient Boosting achieve very good predictive results and consistently outperform standard deep neural models on tabular-style datasets where features are individually meaningful and do not have strong multi-scale temporal or spatial structures [41]. On the other hand, deep learning models are more appropriate in fields like image recognition, speech recognition, and natural language processing.
A balance of computational efficiency, ease of use, and high accuracy have made tree-based models one of the most popular non-linear model types.

Random Forests

Random Forest models overcome the Decision Tree trade-off between overfitting and accuracy. They do so by creating a huge number of D_{train} bootstrap replicas, i.e. datasets of the same D_{train} size composed of D_{train} units sampled at random with repetition. On average these datasets contain only about 67% of the original units, while some of them are duplicated and others are not included. On each bootstrap replica, a full-grown Decision Tree is trained and the predictions of each model are averaged together. In addition, Random Forest consider only a handful of variables as candidates for the split, at each Tree iteration.

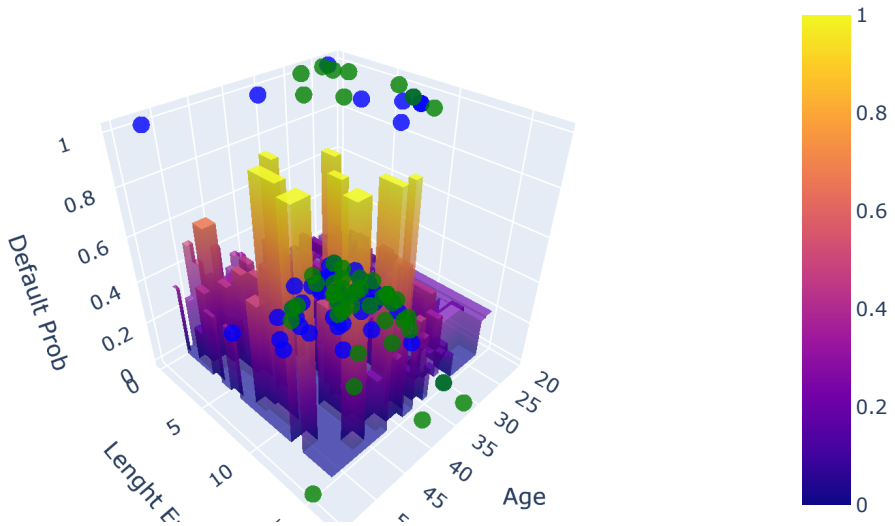


Figure 2.7: Decision Tree model surface obtained on the Toy Credit Scoring Dataset (Chapter 3.4.1). We notice the piecewise constant f generated over $\mathcal{X}$ by a Decision Tree model.

From a mathematical point of view, the main idea of the procedure is to reduce the Variance of the resulting f function -*namely reducing the overfitting*- while retaining the high accuracy of full-grown trees. In fact, the variance is defined as:

$$\text{VAR}(f(x)) = \frac{1}{k}s + \frac{k-1}{k}c$$

where k is the number of Trees included in the ensemble, s is the variance of the prediction errors made by the single Trees and c is the average correlation of different Tree errors. The mechanism of aggregating different models trained on generated bootstrap replicas of the data is called Bagging (Bootstrap Aggregating), It works on the first term of the formula, reducing the variance by increasing the number k of Trees concurring to the prediction. The Random Forest technique consists of Baggging with the additional randomization of the splitting variable. It targets also the second term, creating more diverse Trees -*because of the restrictions on the splitting variables*- and achieving more uncorrelated Tree errors. In general, it is safe to say that Bagging and Random Forest models are always beneficial, while in the worst-case scenario they leave the Variance of the original Tree untouched. But it is very often the case that they improve the base Tree model accuracy by a healthy amount.

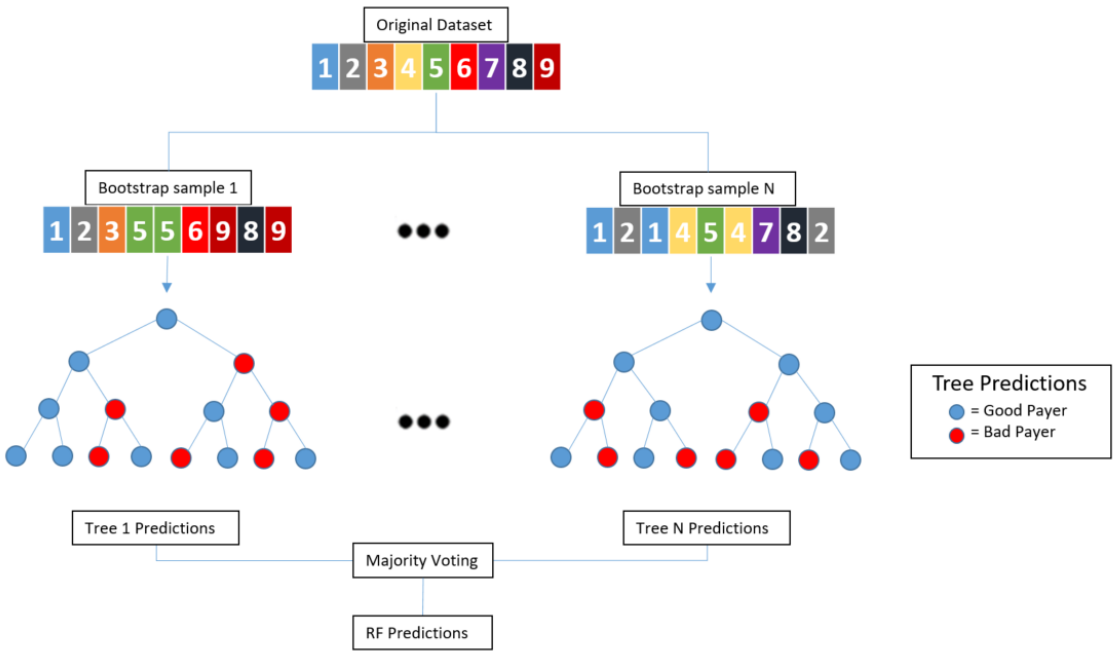


Figure 2.8: Random Forest Structure

From a geometric point of view, the combination of piecewise constant functions produces yet another piecewise constant model, although its complexity is much higher and it displays increased flexibility and robustness -*less overfitting*- than the vanilla Tree model.

Gradient Boosting

Gradient Boosting Models rely on the idea of creating many simple and weak models, also called learners, sequentially added into an Ensemble Model. Shallow Tree models are one of the most popular choices for weak learners. Each Tree is grown on the same D_{train} dataset, but each unit is assigned a different weight at every iteration. The weight is based on the prediction error of the ensemble model built so far. Thereby, units which are already predicted well are given low weights, whereas individuals presenting imprecise or wrong predictions will benefit from higher weights. This allows the following trees to focus more on the hard-to-predict individuals.

On one hand, this approach allows to create an ensemble model, able to predict well also complex and highly non-linear parts of the regressors' hyperspace $\mathcal{X}$. On the other hand, the ensemble is prone to overfitting and a D_{val} validation dataset is required to check the generalization performance.

Weak Learners are simple models achieving modest performance (usually their accuracy is just above chance), such as Single Decision Trees with very few branches. They are employed in boosting procedure, instead of strong learners, because this helps the algorithm to learn "slowly": small performance improvements are made per each shallow Tree added to the ensemble. Slow convergence towards the best f function allows stopping the

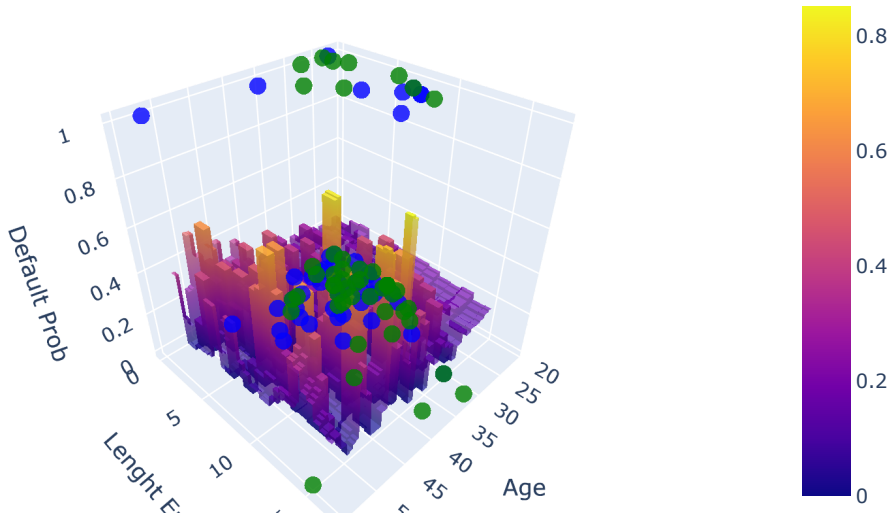


Figure 2.9: Random Forest surface obtained on the Toy Credit Scoring Dataset (Chapter 3.4.1). We notice that the Random Forest surface is more balanced and contains smaller bumps (less f Variance) than Decision Tree model in Figure 2.7

procedure before incurring overfitting. It increases the final model robustness and helps also to keep f simple, e.g. when Decision Trees with just one split are employed, namely stumps, the Boosting final model can be regarded as an additive model [61].
The best single tree at each step is selected minimizing the loss function using gradient descent, hence the name Gradient Boosting. The framework steps are detailed in Figure 2.11.

2.4.0. Generative Models

According to a Gartner study [13], 60% of all data used in the development of AI will be synthetic rather than real by 2024. Such prediction perfectly highlights the exponential usage of Generative models to produce Synthetic data, as well as its huge potential. It is therefore important to have a good understanding of the techniques, to be able to analyze their behaviour and the implications related to current regulations, especially in relation to the Explainability quest.

Generative models learn an approximation of the joint distribution, $D(X, Y)$, instead of learning the conditional distribution $D(Y|X)$ [87]. Having access to a good approxima-

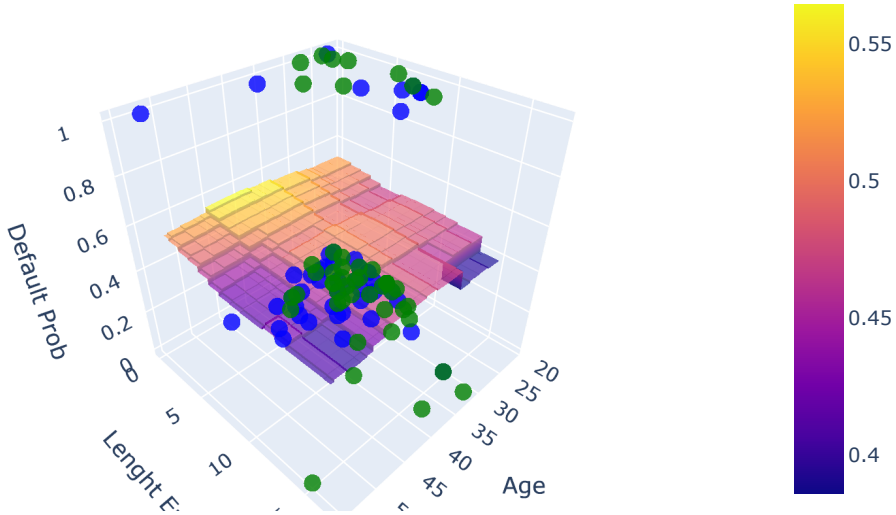


Figure 2.10: Gradient Boosting surface obtained on the Toy Credit Scoring Dataset (Chapter 3.4.1). The surface is usually smoother than Random Forest, maintaining increased robustness and accuracy than vanilla Decision Trees

tion of the joint distribution, it is possible to generate new data maintaining the same statistical properties of the original one, i.e. Synthetic Data.

The first attempt to build a generative model dates back to Donald Rubin in 1993, which proposed multiple imputation to generate synthetic observations of the US Census Bureau [97].
Since then, many different generative model frameworks have been proposed, with Variational Autoencoders (VAE) and Generative Adversarial Networks (GANs) being recognized as state-of-the-art.

Generative Adversarial Networks (GANs)

Adversarial Learning is a framework composed of two separate models, the Generator G and the discriminator D. Both are mathematical functions: $G : \mathcal{Z} \to \mathcal{X}$ converts randomly generated observations over $\mathcal{Z}$ in observations belonging to the same euclidean space of the training data, i.e. $\mathcal{X}$. On the other hand, $D : \mathcal{X} \to [0, 1]$ computes the probability that an element in the $\mathcal{X}$ space is a real D_{train} unit (opposed to being fake,

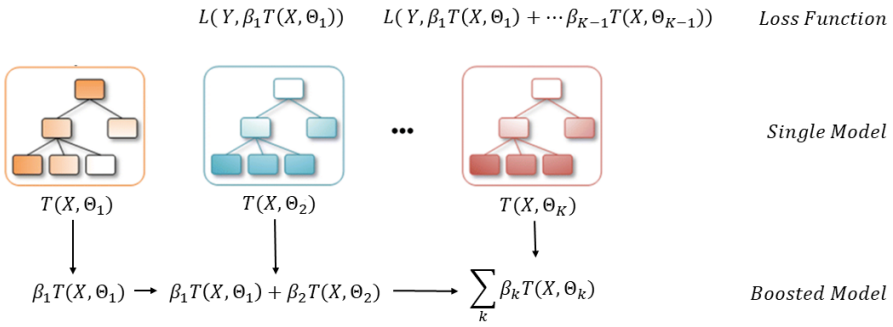


Figure 2.11: Gradient Boosting Tree Model construction.

$T(X, \Theta_k)$ is the best Tree built at step k, its parameters Θ_k are chosen to minimize the Loss Function between the target variable Y and the Boosted Model of the previous step. The Tree is added to the Boosted Ensemble weighted by the β_k parameter.

created by the G function). Specifically, the model focuses on the value function:

$$V = \mathbb{E}_{\boldsymbol{x} \sim D_{train}}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}}[\log(1 - D(G(\boldsymbol{z})))]$$

D objective is to maximize V, while G aims at minimizing $\max_D V$, hence the GAN loss function corresponds to:

$$\min_G \max_D V(D, G)$$

In GANs [35], both G, D are modelled as Neural Networks, specifically as MLP. This is not the only possible choice, but it is a convenient one since allows iterative training of both models at the same time, by updating the parameters according to the error gradients through backpropagation.
Even if it is relatively easy to compute and update parameters according to the gradients, we have to solve a minimax optimization and mathematical tools for this task are still considered immature. Block Coordinate Descent is usually employed in order to train GANs. It trains iteratively one model at a time alternating between D and G, helping both models to improve performances by challenging themselves to higher goals, step by step. Although, it is still quite complex to train GAN models and often the training phase might get stuck into issues such as: vanishing gradients (which prevent G performance improvement), model collapse (when G learns only very specific paths to fool D and the resulting generated observations are very similar) and few more.
Nonetheless, extraordinary results have been obtained through GANs, outperforming Variational AutoEncoders especially in structured data domains such as Text, Images

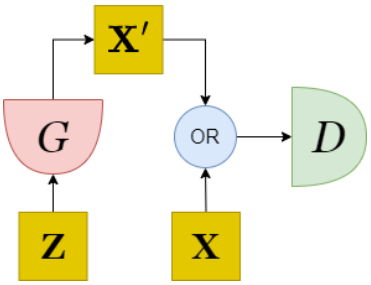


Figure 2.12: Generator G is trained to map a noise sample Z to synthetic data X'. Discriminator D is trained to distinguish real data X from synthetic samples

and Videos. Examples of successful applications are: GANs able to create artworks [31] or images and videos so vivid to be confused with real ones (Figure 2.13).

Figure 2.13: Non-real images generated by GANs in [11], also called DeepFakes

Variational AutoEncoders

Autoencoders [70] are a specific neural network architecture, with input and output layers of the same size. The model is trained using the Reconstruction Loss, i.e. minimizing the difference between input and output. The main goal is to obtain a D_{train} representation in a (usually compressed) latent space $\mathcal{Z}$, represented by the middle layer of the network. The Encoder, i.e. the left side of the network, encloses the function $h : \mathcal{X} \to \mathcal{Z}$ which transforms the data into their dual representation in the latent space. The Decoder, right side of the network, encodes the inverse transformation function $h^{-1} : \mathcal{Z} \to \mathcal{X}$.

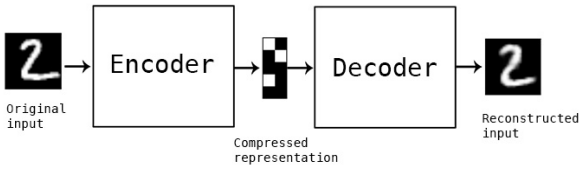


Figure 2.14: AutoEncoder structure

In order to generate data, we sample points on $\mathcal{Z}$ and use the Decoder h^{-1} to obtain Synthetic Data D_{synth} living in $\mathcal{X}$. In AutoEncoders, in fact, the Decoder h^{-1} is regarded as the actual generative model. Unfortunately, we have no clue on the best strategy to sample from $\mathcal{Z}$ -*we did not force any specific distribution on* $\mathcal{Z}$ *during the training phase*-. It

is hence hard to ensure that the generated points are meaningful.

To the rescue, it comes the Variational AutoEncoder [68] technique, which works on the posterior distribution of the latent space $p_{z|x}$. Given an x point and knowing the posterior $p_{z|x}$, we could generate synthetic points similar to x, by sampling z values from the more likely regions of $p_{z|x}$ -*the* $h^{-1}(z)$ *points should highly resemble the original* x *points*-. Since computing $p_{z|x}$ is hard, VAE resort to Variational Inference approximating $p_{z|x}$ with a convenient distribution $q_{z|x}$. The goal is to minimize the difference between the p, q conditional distributions, by minimizing their Kullback-Leibler divergence [71]. It can be proved that minimizing the KL divergence among p, q is equivalent to maximising the Evidence Lower Bound (ELBO) $\mathcal{L}$, such as

$$\min_{q,z} KL(q_{z|x} \| p_{z|x}) \equiv \max_{q,z} \mathbb{E}_q[\log p_{x|z}] - KL(q_{z|x} \| p_z)$$

where $\mathcal{L} = \mathbb{E}_q[\log p_{x|z}] - KL(q_{z|x} \| p_z)$ is composed of two very different quantities: the expected log-likelihood of x given z, which relates to the Reconstruction Loss -*the higher, the better the reconstruction of* $h^{-1}(h(z))$- and the KL divergence between the approximate posterior $q_{z|x}$ and a prior p_z which is usually the standard Gaussian $\mathcal{N}_{\mathcal{Z}}(0, I)$. The latter term ensures to obtain well-behaved distribution of the transformed training data $h(x)$ into the latent space $\mathcal{Z}$.

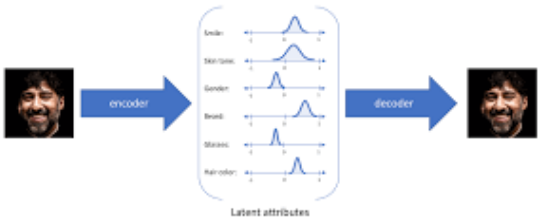


Figure 2.15: Variational AutoEncoders represent each variable of the latent space with a distribution tending to a standard Gaussian. Courtesy of Jeremy Jordan

The two goals are contrasting each other, and the best generative solutions are obtained by finding the proper balance with the help of a tunable hyper-parameter β controlling the trade-off:

$$\mathcal{L} = \mathbb{E}_{q_{z|x}}[\log p_{x|z}] - \beta KL(q_{z|x} \| p_z)$$

Eventually, the reparametrization trick allows us to train the VAE network through back-propagation even if it involves a sampling procedure. In fact, we generate synthetic data sampling from $\mathcal{N}(0, 1)$, then we scale the points into $q_{z|x}$ domain by adding μ and multiplying by σ. Decoupling the sampling step and the usage of the parameters allows us to backpropagate the loss error through the network.

Chapter 3

Datasets

We are going to introduce the datasets, along with specific Machine Learning models trained on them. The dataset, model pairs are used throughout this thesis to demonstrate some of the claimed and already known results about Explanation methods, as well as to test performances and quality of specific Explanation frameworks.

3.1.0. Crif Credit Scoring dataset

The dataset presented here comes from a real-life loan application process, obtained by pooling several anonymised statistical samples from different Italian financial institutions. It contains several demographic, economic and financial variables used as predictors, whereas the target variable consists of the default of the borrower person. However, payment history changes case-to-case and we must come up with a default definition. We consider "bad payers" as: users with 90 or more days past due for at least one payment towards the bank, or individuals with at least one shift from a past due to an actual loss in the last 12 months. We deal with a classification problem, where individuals are grouped into two classes: class 1 if the default occurred (bad payer definition), 0 otherwise. Since only two values are allowed, the variable is said to be binary. Prediction Models estimate the Probability of Default (from now on PD), which assumes any continuous value in the range $[0, 1]$.
The dataset composition is shown in Table **??**.

We split randomly the dataset into two non-overlapping samples: D_{train}, consisting of 70% of the entire dataset, is employed to tune the algorithm; whereas D_{test}, composed of the remaining 30% of the observations, is useful for checking the algorithm's performances on new data.
The first step has been to select the most important 20 features, by screening variables out based on high correlation and testing that the model performance didn't significantly decrease without the irrelevant variables. This is done for two reasons: (i) classical models are not performing well in high dimensional settings and (ii) LIME applied to high dimensional Machine Learning models would cause the method to fail. Subsequently, we apply tree-based Machine Learning models, specifically Gradient Boosting Trees (see Chapter 2.3). They retain the enhanced predictive power of Machine Learning models,

Table 3.1: NHANES Dataset Composition.

dataset	dimension	%Bad
Train set	39.418	2,9%
Test set	16.893	3,1%
Total	56.311	3%

while having the additional advantage of requiring almost no pre-processing, at the same time they cope with outliers and extreme values easily. For a comparison of performances and potential between Logistic Regression -*gold standard in Credit Scoring*- and Gradient Boosting Trees, we refer the reader to Visani et al. [117].
Gradient Boosting's hyperparameters have been tuned by means of grid search and 10-fold cross-validation on the training set.

3.2.0. NHANES Dataset

We introduce another real-world dataset coming from the medical domain. In Medicine, diagnostic computer algorithms providing accurate predictions have countless benefits, notably they may help save lives as well as reduce medical costs. However, precisely because of the importance of these matters, the rationale of the decisions must be clear and understandable.

The dataset is usually referred to as NHANES I and was originally described in [19]. It has been employed for medical research [24],[74] as well as a benchmark to test explanation methods [79].
We use a reformatted version, released at http://github.com/suinleelab/treexplainer-study. It contains 79 features, based on clinical measurements of 14,407 individuals. The aim is to model the risk of death over twenty years of follow-up.

The data have been divided into a 64/16/20 split for train/validation/test following Lundberg et al. [79] prescriptions (Table 3.2 for details). The features have been mean-imputed and standardized based on statistics computed on the training set. A Survival Gradient Boosting model has been trained, using the XGBoost framework [16]. Its hyperparameters have been optimized by coordinate descent, using the C-statistic [50] on the validation set as the figure of merit, obtaining a sound 0.82 out of 1 for the final model. The model prediction consists of the hazard ratio for each individual: higher prediction means the individual is likely to be alive for a shorter period of time. Therefore, positive variables coefficients define risk factors, whereas protective factors have negative values.

The dataset is used throughout the thesis to demonstrate how the techniques hereby presented are widely applicable to other Tabular domains. NHANES is mainly employed to explain how LIME prediction can come in handy (Chapter 4.2) and to show an OptiLIME application in Chapter 7.3.

Table 3.2: NHANES Dataset Composition

dataset	dimension
Train set	9.221
Validation set	2.305
Test set	2.881
Total	14.407

3.3.0. Real-World UCI Datasets

To test different methods and techniques, we exploit real-world datasets freely available from the UCI repository. We use three different datasets, whose properties are summarized in Table 3.3. We considered datasets with continuous features, in order to simplify the adoption of different ML models and explanation techniques, and with increasing dimensionality to be able to appreciate differences in performance due to the number of features involved. The ***Wine*** dataset[1] [18] contains physical measurements for different wines. The task is quality prediction, which we consider as a regression task. The ***House sell*** dataset[2] contains house sell prices and other information related to the sale in King county (Seattle greater area). Here the task is price prediction. The ***Parkinson telemonitoring*** dataset[3] [112] contains data from patients with early-stage Parkinson's disease. The goal here is to predict a symptom score on a normalized scale.

These real-world datasets are mainly used to prove the soundness of LIME Stability Indices in Chapter 6 as well as to quantitatively compare different explanation methods on both local and global explanations in Chapter 8.1.4.

Table 3.3: Datasets description.

dataset	dimension	n_{train}	n_{test}
wine	12	4,163	735
house	19	18,371	3,242
Parkinson	26	4,993	882

On each of these datasets, we apply three different models, trained similarly across all datasets. The first is an ***XGBoost*** model [16]. XGBoost iteratively aggregates base regressors greedily minimizing a proxy loss. Following Friedman [28] prescriptions, we employ simple CART trees with maximum depth 2 as base regressors, learning rate of 0.05, and early stopping rounds set to 100. The second is a ***multi-layer perceptron*** (MLP), composed of two hidden dense layers of 264 neurons each, trained by adaptive stochastic

[1]available at https://archive.ics.uci.edu/ml/datasets/wine+quality
[2]available at https://www.openml.org/search?type=data&status=active&id=42092
[3]available at https://archive.ics.uci.edu/ml/datasets/parkinsons+telemonitoring

gradient descent [67, ADAM] and early stopping on a hold-out validation set. Both these models achieve state-of-the-art accuracy in many settings and are not inherently interpretable. The thrid model is a vanilla ***Linear Regression*** trained using the scikit-learn implementation [90]. No regularization has been applied and default parameters have been used. The training is done by Gradient Descent.

3.4.0. Toy Datasets

3.4.1.0. Credit Scoring Toy Dataset

The dataset [113] comes from the Kaggle Repository. It describes a Loan application process and contains information regarding the applicants, such as *Age, Annual Income, Home ownership, Employment length,* etc.
After removing missing values and outliers, the dataset counts 28.621 individuals and 12 recorded variables.

Throughout the thesis, the dataset is mainly used to produce insights on Machine Learning models and Explanation techniques, applied to the Credit Scoring field. It acts as a replacement for the real-world Credit Scoring dataset of Chapter 3.1, which is Crif proprietary, hence it was not possible to use it for all kinds of analyses, due to privacy obligations towards customers.

A small snapshot of the dataset has been already introduced in Table 1.1.

3.4.2.0. Synthetic one-dimensional Dataset

In order to provide a better understanding of the inner working of the LIME algorithm and the proposed modifications, we introduce a very simple univariate dataset, generated from the following Data Generating Process:

$$Y = sin(X) * X + 10$$

100 distinct points have been generated uniformly in the X range $[0, 10]$ and only 20 of them were kept, at random. In Figure 3.1, the blue line represents the True DGP function, whereas the green one is its best approximation using a Polynomial Regression of degree 5 on the generated dataset (blue points). In the following we will regard the Polynomial as the ML function f, we will not make use of the True DGP function (blue line) which is usually not available in practical data mining scenarios. The red dot is the reference point in which we are going to evaluate local explanations. The dataset is intentionally one-dimensional, so that the geometrical ideas about LIME may be well represented in a 2d plot.

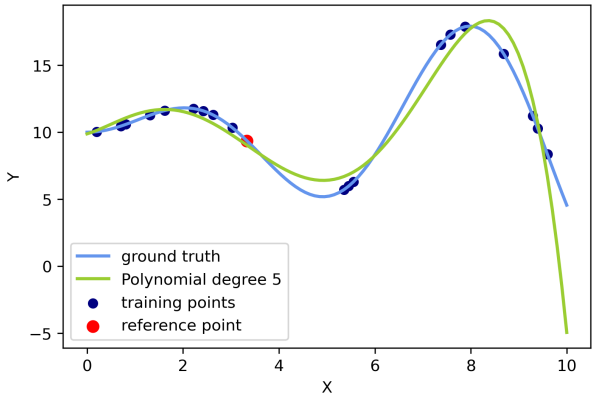


Figure 3.1: Toy Dataset

Chapter 4

Explanations

Recalling Hall and Gill [46], ***Interpretability*** can be regarded as "the ability to explain or to present the results [of a prediction system], in understandable terms, to a human". The concept ***human-understandable*** is an inherently subjective matter which is hard to quantify. However, even if we have no rigorous way to compute interpretability, it is clear that the complexity of some prediction frameworks goes beyond average human comprehension, requiring tools to understand their behaviour. We call ***Explanation*** any generic tool of this sort.

Rigorously defining an ***Explanation*** is equally hard, since it is a direct consequence of the ***Interpretability*** concept. Quite a few proposals have been laid out in recent years [66, 94, 105], but no consensus has yet been reached on the formal definition. Turning to the various frameworks recently proposed, we recognize that an ***Explanation*** can be conveyed by means of text, visual or numerical outputs, or a mix of them. They can be general tools or tailored to specific users and applications [96]. The whole collection of methods and techniques is usually referred to as the Explainable Artificial Intelligence (XAI) field.

XAI approaches can be grouped based on different criteria such as i) Model agnostic or model specific ii) Local or global iii) Intrinsic or post-hoc iv) True to the black-box Model or True to the Data [82, 44, 15].

Intrisic vs Post-hoc

Intrinsic interpretability refers to prediction models with a relatively simple structure, entitling them to be considered explainable by default. It is usually achieved by restricting the complexity of the model surface, as in the Parametric models approach.

On the other hand, ***Post-hoc*** explanations aim to explain the model's predictions retroactively. They can be seen as a separate additional layer which can be applied at any time after training. ***Post-hoc*** methods can also be applied to intrinsically interpretable models.

Model-specific vs Model-agnostic

Model-specific interpretation tools are limited to specific model classes and they are often based on the properties of the model, such as its architecture, the training data and algorithm used, as well as its underlying assumptions and biases. By definition, interpreting intrinsically interpretable models is always ***model-specific***, for example interpreting the regression weights in a linear model. Similarly, tools that only work for the interpretation of a given class of complex black-box models e.g. neural networks are ***model-specific*** as well.

Model-agnostic tools are not specific to any particular model, and they aim to provide explanations that apply to a wide range of prediction tools. As such, ***model-agnostic*** techniques are applied after the model has been trained (in a ***post-hoc*** fashion). These agnostic methods exploit the model f surface lying in the geometrical space $\mathcal{X}, \mathcal{Y}$ and they provide insights on that. Since each prediction model generates the f surface, such methods are generalizable to any class of ML and Statistical models. However, it is strictly forbidden for them to use the model internals such as weights or structural information: this would cause the explanation framework to rely on the programmatic characteristics of a specific model class, hindering its ability to generalize to other classes.

Local vs Global

Recalling the notion of the model f surface, ***Local*** techniques focus on a (usually) small subregion of $\mathcal{X}$ and provide insights on f, restricted to that sub-region. They generally consider a specific input unit and define a neighbourhood around it. Few frameworks clearly state the neighbourhood boundaries, while the majority of the techniques specify them implicitly. As such, it is hard to know exactly where local explanations still provide valid insights on f.

Global explanations, on the other hand, target the entire $\mathcal{X}$ space on which f lives. Their goal is to provide insights related to the model as a whole. This task is clearly harder than the Local setting one, since f can show a high degree of complexity on $\mathcal{X}$. It is instead reasonable to consider that local and small sub-regions of $\mathcal{X}$ limit f complexity to a lower degree, making the local insights indeed easier to retrieve. At the same time, Global explanations are highly desirable, since they provide a global and full understanding of the black-box model. Local explanations may sometimes fall short of generalization capabilities, making them not very suitable under current AI regulations.

True to the Model vs True to the Data

Chen et al. [15] put forward a very interesting point of view relative to whether explanations should be ***True to the model*** or ***True to the data***. Considering again the X dataset geometric viewpoint as a cloud of dots in the $\mathcal{X}$ space (Chapter 2), the problem can be related to different densities of the points of the dataset among different regions in the data manifold. In particular, specific patterns in the data might cause some regions to be empty. As an example, it is not possible for a 14 years old teenager to hold a valid driving license, hence the region defined by "Age $<$ 18" and "Holding Driving License

= True" would contain no dataset units. Highly correlated variables have an effect on the data distribution over $\mathcal{X}$ as well: eg. it is very unlikely to find individuals very tall (say 200 cm) but extremely light (50 kg). Hence, the specific region would have very low density.

True to the model explanations aim at providing insights about the model surface, considering each part of the $\mathcal{X}$ space as equally important (also regions of low or null data density). The focus is on the f model surface and not on the available data.

On the other hand, the ***True to the Data*** approach proposes to provide weighted insights on f, i.e. low density regions will be taken into account with a reduced emphasis, based on their data distribution density.

Chen et al. [15] argue that no framework is clearly superior to the other, instead the choice is application dependent. As an example, the ***True to the Model*** approach is desirable in a Credit Scoring scenario, when explanations are used to suggest how rejected applicants should modify their behaviour to receive a loan. The focus here is on the specific model used by the bank and we are interested in each possible change, i.e. consider any region even if it has low probability.
Consider now the goal to understand which genes' expressions determine a particular outcome (e.g., response to anti-cancer drugs), by measuring gene expression in a set of patient samples and applying prediction models on this data [128]. The focus is now on the patients and their response to drugs. We value a drug as more effective if its reaction can be observed in a great number of patients, while effects on a very specific group of patients only, with limited coverage in the clinical trial, are not highly interesting. Low-represented regions do not bear high importance, hence the ***True to the Data*** framework is preferable.
Usually, the ***True to the model*** approach is required when the model needs to be compliant with legal requirements and regulations. In such situations, the model itself is the focus and we want to ensure that it is trustworthy in any part of the manifold.

There is still no gold standard for explanation methods: each class presents pros and cons and on top, each technique has its specific perks and downsides. In the following, we are going to focus on ***Post-hoc model-agnostic explanations***, specifically tailored ***for Tabular Data***. In general, model-agnostic methods for Tabular data rely on the geometrical interpretation shared by any prediction model -*Chapter 2*-. It is possible to provide both local and global insights on the multivariate shape of f, as well as to focus on the data distribution or the surface of the f model itself. Therefore, the techniques explained here belong to any combination of the Local/Global, True to Model/Data properties.

The remainder of the Chapter will present different techniques grouped according to two different ways of producing explanations, i.e. Feature Attribution Techniques in Chapter 4.1 and Surrogate Models in Chapter 4.2. To conclude we also present different methods to assess the quality and validity of the retrieved explanations, in Chapter 4.3.

4.1.0. Feature Importance Techniques

The first group of techniques share the concept of ***feature importance*** or ***attributions***, i.e. a single number summarizing the contribution of each feature. The ***attributions*** can be used to rank the variables, which in turn may be beneficial to understand which variables are not used by the trained prediction model and can be safely excluded in a feature selection step to reduce the model complexity. Hereafter, we introduce the Function Decomposition framework, which provides an innovative viewpoint to analyze ***feature attribution*** methods. Subsequently, we shed light on the state-of-the-art explanation techniques belonging to this class.

Function Decomposition

Any multivariate function can be additively decomposed, without losing any information, in main effect terms and interaction terms of any order (from 2 to d).

$$\begin{aligned} f(X) =& f(X_1, \cdots, X_d) = \\ =& f_1(X_1) + \cdots + f_d(X_d) + f_{1,2}(X_1, X_2) + f_{d+2}(X_1, X_3) + \cdots \\ &+ f_{d-1,d}(X_{d-1}, X_d) + f_{1,2,3}(X_1, X_2, X_3) + \cdots + f_{1,\cdots,d}(X_1, X_2, \cdots, X_d) \end{aligned}$$

Each $f_{\cdot}$ function encodes only the specific variables involved. Moreover, each interaction term accounts only for the interaction of the variables of that specific order, eg. $f_{1,2,3}(X_1, X_2, X_3)$ models only the three-way interaction of the variables, it does not comprehend any two-way interaction effect as $f_{1,2}(X_1, X_2)$ or any higher order interaction coded instead by $f_{1,2,3,4}(X_1, X_2, X_3, X_4)$ for example. In a similar fashion, main effect terms do not include any interaction effect.
Prediction models make no difference and the function decomposition applies to any of them.

The Decomposition Rule states that for any function f there exists a suitable decomposition, given by $f_1, \cdots, f_{1,\cdots,d}$. Although the f_i decomposition functions are generally unknown -*apart for trivial situations, such as the linear regression framework in which f is already in the decomposed form*-. Usually, we don't know the functional form of any main effect or interaction in Machine Learning models. Just to give an example, an interaction term $f_{1,2}(X_1, X_2)$ may be encoded by $f_{1,2} = X_1 X_2$, $f_{1,2} = \log(X_1) X_2$ or $f_{1,2} = \frac{X_1}{X_2}$ or infinitely many other shapes.

In general, a function f_j is 0 when the single variable (or the combination of variables) is not needed to recover f. On the other hand, when f_j presents non-zero values for at least a compact interval of the X_j domain, we consider it meaningful in the f decomposition. It is also possible for a variable X_j to have zero-valued main effect f_j but meaningful interactions with specific variables, eg. $f_{j,i}$. This would mean that the variable has an impact only when combined with specific X_j values.

We consider such a viewpoint, put forward by Friedman and Popescu [29], as extremely useful to understand the different Explainability Methods based on Feature Attribution.

Drop-Column Importance

A popular approach is to exclude a certain feature, or group of features, from the model and evaluate the loss incurred in terms of model goodness. Such value quantifies the importance of the excluded feature: a high loss value underlines an important variable for the prediction task.

There exist different ways for excluding the feature from the model f. The practitioner can use the ***Drop-column feature Importance*** by simply removing the column related to the specific feature j from the dataset, and retraining the model. She would obtain a new different model $f_{\setminus j}$.
The expected value of the difference in prediction between $f, f_{\setminus j}$ is computed on the data, using the sample mean as approximation:

$$\text{Imp}_j = \frac{1}{n}\sum_{i=1}^{n}(|f(x^{(i)}) - f_{\setminus j}(x^{(i)})|)^p \tag{4.1}$$

where p is a constant chosen by the practitioner a priori and $f_{\setminus j}$ represents the model trained on the D_{train} data without the j-th variable. Another option is to use an aggregate measure of performance as a proxy of Imp_j, based on the nature of f. Just to give the flavour, few suitable aggregate metrics are: AUC (Area Under the ROC Curve), Accuracy, F1 for classification models; MSE (Mean Squared Error), RMSE (Root Mean Squared Error), MAE (Mean Absolute Error), MAD (Mean Absolute Deviation) for regression models.

The Drop-column technique is ***highly computationally*** intensive since it requires training d different models (one per each variable). In addition, model $f_{\setminus j}$ is generally different from the marginalization of f over the j-th variable. In fact, re-running the optimization algorithm on the modified D_{train} *-without the j-th variable-* is likely to retrieve a different stationary point *-local minimum-* of the Loss Function, which corresponds to a different model surface f' in $\mathbb{R}^d$. All in all, this means that we are comparing performances of two different models on the same D_{train} dataset, hence using a ***True to the data*** approach.

Permutation Importance

Permutation Importance is an alternative and less expensive way to compute attributions, based on the idea that a feature is "unimportant" if shuffling its values leaves the model error unchanged. It was first introduced by Breiman [9] specifically for the Random Forest model and has been generalised to a model-agnostic framework, named LOCO [75].
The technique relies on randomly shuffling the j-th feature to obtain a new dataset $D^{(j)}_{train}$, and compute Eq. 4.1 or the difference of any aggregate performance measure among $f(D_{train})$ and $f(D^{(j)}_{train})$.

The Permutation Importance algorithm is ***faster*** and is guaranteed to ***work on the same original*** f. Although, permuting feature j breaks correlations and changes interactions of any order among feature j and another feature or group of features. The decrease in performance hence considers the full array of f. functions including the j-th

feature of the Function Decomposition, in particular any possible interaction of the feature j with any other feature is fully included in Imp_j. Permutation Importance can ***overestimate attributions***, by not taking into account that the interaction terms share their importance among the different variables involved.

Both Permutation Importance and Drop-Column Importance are generally regarded as global explanations, since the Imp_j quantity is computed as a difference of global performance metrics (over the entire data domain). It is possible to use a local performance metric, which takes into account the units included in a specific subset only, achieving local explanations. To do so, it is however required some form of locality knowledge -*boundaries of the local neighbourhood must be set beforehand*-, not provided by these methods.

Partial Dependence Plots (PDP)

The technique aims at isolating the effect of a specific variable -*or group of variables*-, to understand how it impacts the model f. The core concept is substantially the opposite of Permutation and Drop-Column Importance, but they share the same goal.

Consider splitting the $X_1, \cdots, X_d$ variables in two subsets X_S ,X_C where the set S comprises the variables we are interested in -*the ones we want to isolate*-, C includes the other variables. Friedman [28] proposes to marginalize the model f wrt the set X_C :

$$\hat{f}_S(x_S) = E_{x_C}[f(x_S, x_C)] = \int_{\mathcal{X}_C} f(x_S, x_C)\, p(x_C)\, dx_C \tag{4.2}$$

The function $\hat{f}_S$ depends only on the variables X_S . It tells us for a given value of features S what the average f prediction is. PDP can be thought of as a way to simplify the model, losing some detail but gaining many insights on how the chosen variables affect the model.
In practical terms, we approximate the Expected Value of Eq. 4.2 with the sample mean over D_{train} :

$$\hat{f}_S(x_S) = \frac{1}{n} \sum_{i=1}^{n} \hat{f}\left(x_S, x_C^{(i)}\right) \tag{4.3}$$

Hence the quality of PDP explanations depends on the D_{train} quality -*if some specific patterns are missing in* D_{train} *, the PD function will be distorted as well*-. To implement marginalization in practice, we apply the steps detailed in Algorithm 1. Each loop iteration computes one point of the $\hat{f}_S(x_S)$ function, which can be subsequently plotted for each variable separately, as in Figure 4.1. High variance PD functions highlight important variables, which have a strong impact on the average f prediction.

Partial Dependence Functions are quite intuitive but lack a numeric value to rank variables in order of importance. Greenwell, Boehmke, and McCarthy [38] propose a straightforward extension to obtain PDP attribution scores, which consists in computing the stan-

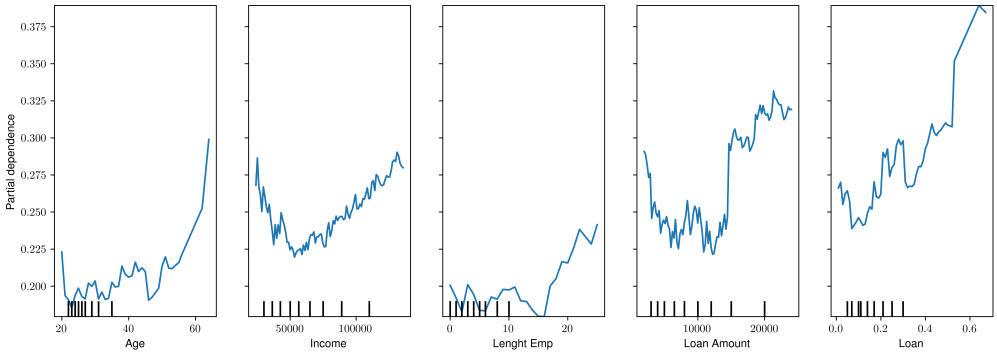


Figure 4.1: PD Plots for single variables on the Toy Credit Scoring Dataset

dard deviation of the PD points:

$$\text{Imp}_S = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(\hat{f}_S\left(x_S^{(i)}\right)^2 - \frac{1}{n}\left(\sum_{i=1}^{n}\hat{f}_S\left(x_S^{(i)}\right)\right)^2\right)} \tag{4.4}$$

PDP has a computational cost of order $\mathcal{O}(dn^2)$), which makes it unfeasible for large datasets. Some expedients to reduce it are: i) evaluate $\hat{Y}$ on a selection of X_C representative points only, ii) sample a reduced number of D_{train} points on which to compute PD, ii) exploit specific structures of the ML models (eg. tree-based models) to speed-up $\hat{Y}$ computation.

Algorithm 1 `Compute PD Function` $\hat{f}_S(x_S)$

1: **for** each observation $x^{(i)} \in D_{train}$ **do**
2: generate new $\hat{D}_{train}$ dataset, with generic unit $\hat{x}^{(j)} = \left[x_S^{(i)}, x_C^{(j)}\right]$
3: predict $\hat{Y} = f(\hat{D}_{train})$
4: **return** Average of $\hat{Y}$

PDP also presents some flaws especially when it comes to ***correlated variables***. In fact, in Step 2 of Algorithm 1 combines any x_C value contained in D_{train} with $x_S^{(i)}$, generating new points. However, there is no guarantee these points are likely to be observed in real data -*new points may lie in regions of low or no* D_{train} *density*-.
Another drawback arises with ***heterogeneus effects***, i.e. the same variable can have inverse impact on different observations. Since PDP employs expectation over the entire D_{train}, some heterogeneous effects may be averaged out and the practitioner will not notice them in the aggregated $\hat{f}_S$ function.

Eventually, considering the PDP formulation (Eq. 4.2) from the Function Decomposition perspective, we notice that taking the Expected Value wrt X_C transforms each $f_{\cdot}$ not

depending on X_S into a constant. On the other hand, any $f_{\cdot}$ containing X_S is not averaged out and concurs to the $\hat{f}_S(x_S)$ shape. This includes the main effects f_S and all the possible interactions $f_{\cdot,S}$ of any order. As an example, let's consider $PD(X_1)$ and $PD(X_2)$: $f_{1,2}$ is fully included in both functions, instead of being distributed among the two. This means that variables interacting with each other have an edge compared to variables having main effects only. We also recognize that PDP is a global method, since it estimates $\hat{f}_S$ over the entire domain of the X_S variables.

Individual Conditional Expectation

Individual Conditional Expectation (ICE), introduced by [34], aims to ***solve*** the ***heterogeneus effects*** problem of the PDP. In fact, ICE exploits the same calculations as PDP, but instead of aggregating f predictions through an average, it records $\hat{f}(x^{(i)})$ separately and generates distinct functions $\hat{f}_{x_S^{(i)}}\left(x_S^{(i)}\right)$, one per each point in D_{train}. Displaying the $\hat{f}_{x_S^{(i)}}$ functions in a single plot, we can get an idea of possible heterogeneus effects which are not averaged out in the ICE framework.

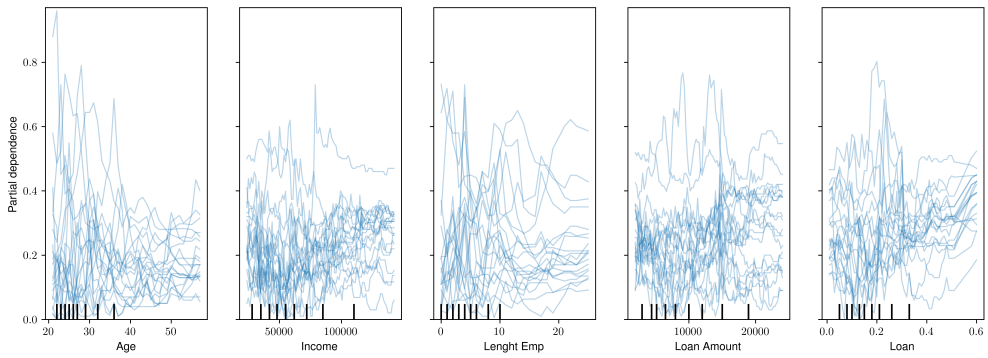


Figure 4.2: ICE plots of the individual effects of the X variables on the Probability of Default, in the Toy Credit Dataset. Each trajectory highlights the PD of a specific unit, changing only the value of the given X_j variable.

ICE comes with the same computational cost of PDP as well as the same drawbacks (apart from the heterogeneous effects being addressed) and the same strongpoints. Notably, ICE is still sensitive to variables' correlation. Also, the intuition under the Function Decomposition framework is the same. ICE's main difference is the scope of the explanation: it can be considered a local explanation since each function explain one single point in D_{train}. At the same time, it takes into account the entire X_S domain, which allows to inspect f behaviour globally along the single X_S axis. We can think of it as an explanation of a single dataset unit along the global domain of a specific variable.

In conclusion, ICE is helpful to uncover heterogeneous effects, but can only be applied on single variables -X_S *including only one variable*-, otherwise the plots become too complex to be useful -*while PDP works smoothly with 2 variables at a time*-.

Marginal plots

As already stated, the PDP biggest hurdle comes with correlated X_S ,X_C features, mainly due to the fact that vanilla marginalization integrates wrt the density $p(X_C)$ which is substantially different from the conditional density $p(X_C|X_S = x_S)$ in correlated environments.
Marginal Plots (M Plots) hence consider marginalization over the conditional density:

$$\begin{aligned}\hat{f}_{x_S,M}(x_S) &= E_{X_C|X_S}\left[f(X_S, X_C)\,|X_S = x_S\right] \\ &= \int_{\mathcal{X}_C} f(x_S, x_C)\, p(x_C|x_S)\, dx_C\end{aligned} \tag{4.5}$$

Since we don't know the $p(x_C|x_S)$ distribution in practice, M Plots propose to use a grid of bins over the X_S domain -*turquoise lines on* X_1 *domain in Figure 4.3*-. The idea is to create equally spaced bins, to simulate a coarse conditioning on the X_S (the narrower the bins, the finer the conditioning). A fraction of the original data $D_{train,k}$ is included in the generic k bin, and we approximate the integral inside the bin using the same rationale as PDP (Algorithm 1) on $D_{train,k}$ datapoints. Repeating the procedure over all the K bins gives us various evaluations of the $\hat{f}_{x_S,M}(x_S)$ function, which can be plotted in the so called M Plot.

The major M Plot benefit is to be able to ***deal with correlated variables***, generating points consistent with the original joint distribution $p(X_S, X_C)$. The method works for uncorrelated variables as well, but the bin partitioning reduces the number of points averaged at each step, hence reducing the accuracy of the final attribution values. It is suggested to be used only when high correlations justify it. Moreover, the conditional distribution approximation gets better with a high number of bins (because we have more granularity in inspecting the $\hat{f}_{x_S,M}(x_S)$ function). At the same time, we must retain a reasonable number of points in each bin to ensure the average is meaningful. It is important to choose a good bin size to avoid any of the two abovementioned issues.
Another severe M Plots drawback is the difficult interpretation of the results: in fact, a high variance M Plot function does not ensure that the X_S variable is truly important. In presence of a high correlation between X_S and another X_j variable, the bins created on X_S are incidentally very similar to the ones we would create on X_j . Therefore, a high $\hat{f}_{x_S,M}(x_S)$ variance might be caused by a strong impact of X_j on f while X_S could have just a negligible effect. An example, taken from the Credit Scoring domain, may help fix the concept: "Consider to predict the Default Probability using, among other variables, the ***Credit card Plafond*** and the amount of the ***highest monthly expense***. Suppose that the highest monthly expense has no effect on the predicted Default Probability, only the Credit card Plafond has. The M-Plot would still show that the highest monthly expense increases the Default Probability, since the Credit Card Plafond increases with the monthly expenses."

Accumulated Local Effect

Accumulated Local Effects (ALE) solve the M Plots issue of correct attributions of strongly correlated variables, by computing the expectation of the f derivative wrt X_S

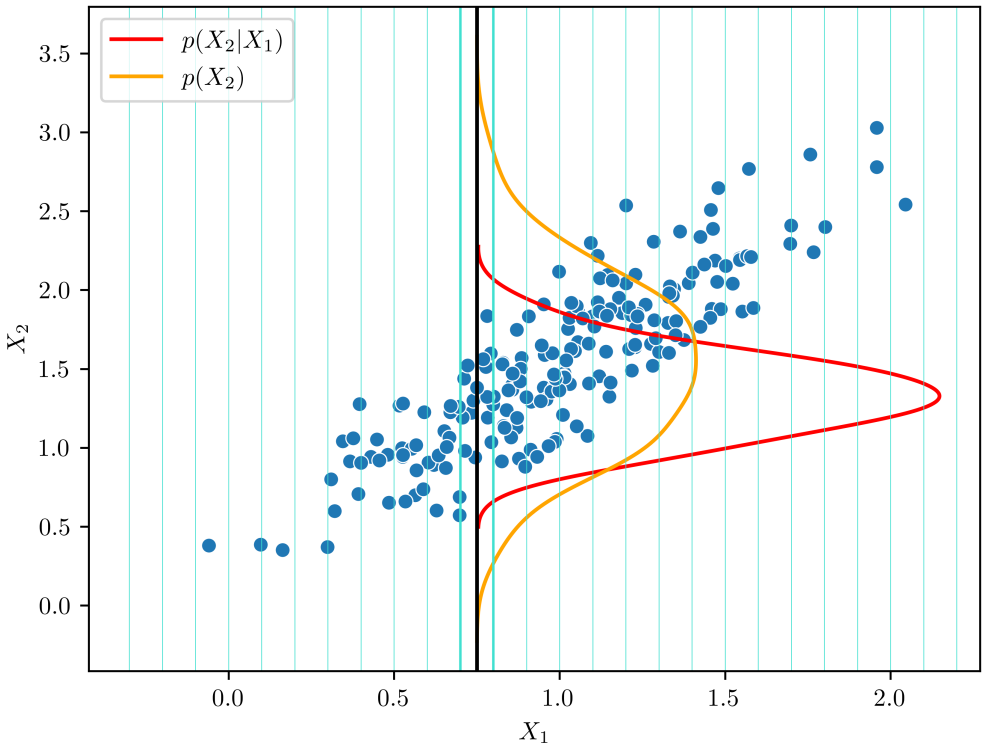


Figure 4.3: X_1, X_2 variables are strongly correlated: $\rho = 0.8$. We need to compute $\hat{f}(x_1 = 0.75)$. In this situation, the Marginal distribution (orange) does not represent the true joint distribution $p(X_1, X_2)$, while the Conditional distribution (red) inside the bin $[0.7, 0.8]$ is representative of D_{train} behaviour.

over the conditional distribution $p(X_C|X_S = x_S)$:

$$\hat{f}_{S,ALE}(x_S) = E_{X_S|X_S=xx_S}\left[\frac{\partial f(X_S, X_C)}{\partial X_S} \mid X_S = x_S\right] \tag{4.6}$$

Conceptually, ALE isolates the effect of X_S by computing the gradient of f wrt X_S. The gradient is approximated locally (using the same bin strategy as M Plots), so the derivative is taken with respect to the conditional distribution.

From an algorithmic point of view (see Algorithm 2), ALE splits the X_S domain into equally spaced bins and considers the left and right boundaries in each bin, x_{S_l} ,x_{S_r} respectively. It generates a data pair $\left(\hat{x}_l^{(i)}, \hat{x}_r^{(i)}\right)$ for each $x^{(i)}$ datapoint inside the bin. The pair has X_C values equal to the original $x^{(i)}$, but the X_S variable assumes value x_{S_l} ,x_{S_r} for $\hat{x}_l^{(i)}, \hat{x}_r^{(i)}$ respectively. Notice that $\hat{x}_l^{(i)}, \hat{x}_r^{(i)}$ differ only in the x_S value, hence a difference in prediction between the two can only be imputed to the X_S feature.
ALE computes $\text{Imp}_{S,k}^{(i)}$ -*difference in prediction*- for each datapoint $x^{(i)}$ falling inside the k bin, and averages them in $\text{Imp}_{S,k}$, the X_S feature attribution in the specific bin.

Algorithm 2 `Compute ALE Plots` $\hat{f}_{S,\texttt{ale}}(x_S)$

1: Split the X_S domain in equally spaced K bins,
2: **for** each bin k (recall data,left,right boundaries: $D_{train,k}$,x_{S_l} ,x_{S_r} **do**)
3: **for** each observation $x^{(i)} \in D_{train,k}$ **do**
4: create $\hat{x}_l^{(i)} = \left[x_{S_l}; x_C^{(i)}\right]$ and $\hat{x}_r^{(i)} = \left[x_{S_r}; x_C^{(i)}\right]$
5: compute $\text{Imp}_S^{(i)} = f(\hat{x}_r^{(i)}) - f(\hat{x}_l^{(i)})$
Compute average $\text{Imp}_{S,k} = \frac{1}{n_k} \sum_{i=1}^{n_k} \text{Imp}_S^{(i)}$
6: store $\text{Imp}_{S,k}\ \forall k$

In order to keep the same visual interpretation of PDP and M Plots, the average gradients $\text{Imp}_{S,k}$ are cumulated in ALE plots. In this way, the plot shows the average f value for each bin instead of the gradient -*which would fluctuate around 0, given increasing/decreasing prediction values imputable to* X_S-. The ALE cumulative behaviour helps understand the impact of X_S on f predictions directly.

ALE can also be computed for variable pairs, by conditioning on both of them. From a geometric point of view, this corresponds to creating rectangular cells over the $\mathcal{X}_{S_1}, \mathcal{X}_{S_2}$ joint domain. Considering the Decomposition Rule, ALE two-dimensional Plots account for both the main effects $f_{X_{S_1}}, f_{X_{S_2}}$ and their interaction term $f_{X_{S_1},X_{S_2}}$.

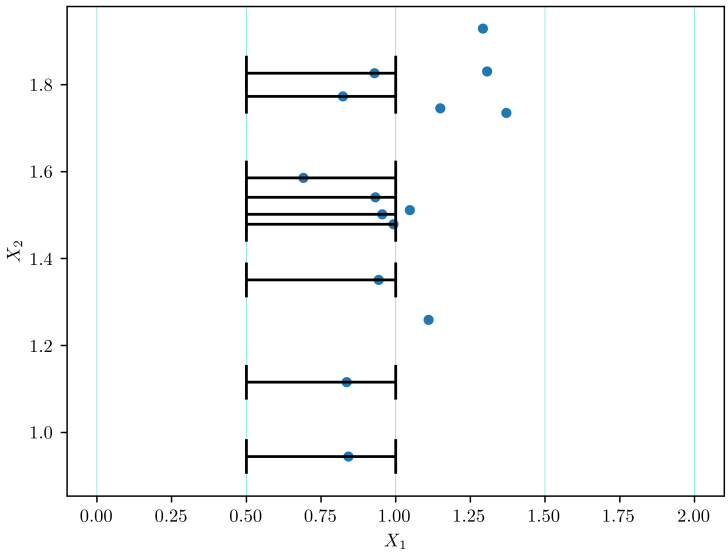


Figure 4.4: ALE computes unit-specific local gradients as the difference in prediction of the projections of the $x^{(i)}$ point on the grid boundaries, namely $f(\hat{x}_r^{(i)}) - f(\hat{x}_l^{(i)})$. The average prediction difference is considered in each bin

ALE's main advantage over PDP and M Plots consists of their ***unbiasedness in highly correlated settings***. At the same time, the binning strategy used to account for the conditional distribution may require some tuning to find the best number of bins allowing for both enough granularity and a reliable gradient estimate inside the bin. With small D_{train} datasets, the technique may become unreliable.
ALE is faster than PDP, since it requires to compute $\mathcal{O}(dn)$ f predictions, precisely only two points need to be predicted for any D_{train} point, for each variable. PDP instead requires $\mathcal{O}(dn^2)$ f predictions.
As far as single variable ALE Plots are straightforward to interpret, multi-dimensional ALE Plots might be more complicated (since they do not separate main effects from interaction effects).

SHAP

The SHAP technique builds on the Shapley Values proposed by Shapley [103] to fairly divide payoffs between players in a coalition, in a cooperative game setting. The Shapley value ϕ_i for a given player i is:

$$\phi_i = \frac{1}{N!} \sum_{S \subseteq N \setminus \{i\}} |S|!(|N| - |S| - 1)![\psi(S \cup \{i\}) - \psi(S)] \tag{4.7}$$

where N is the total number of players, S is any subset of players that can cooperate together, ψ is the payoff function which computes the reward of a given coalition of players. $[\psi(S \cup \{i\}) - \psi(S)]$ is the marginal contribution of player i when added to the coalition S.
We can read the formula as the *average marginal contribution of player i, over all the possible coalitions, starting from the empty coalition up to the grand coalition.*

The Shapley Values are the only way to split the payoff of the grand coalition, respecting the three following properties: i)***Simmetry***, i.e. two players contributing the same amount in any coalition, should have the same reward, ii) ***Dummy Player***, i.e. a player whose marginal contribution to any coalition is the same he achieves alone, should have a ψ as if he was playing alone, ***Additivity***, i.e. Shapley values of two separate cooperative games can be summed to obtain the total reward.

A feature attribution setup can be viewed as a cooperative game: variables in the model cooperate to achieve a more accurate prediction. In particular, selecting a specific unit $x^{(i)}$, we are interested in fairly splitting the prediction value $f(x^{(i)})$ between a baseline -*usually the average* $f(X)$ *value in* D_{train}- and the contribution of each feature.

The Shapley Values formula (Eq. 4.7) translated in a feature attribution scenario becomes:

$$\text{Shapley}\left(X_j^{(i)}\right) = \sum_{S \subseteq F \setminus j} \frac{|S|!|(F - S - 1)|!}{|S|!} L\left(f_{S \cup j}\left(x_S^{(i)} \cup x_j^{(i)}\right) - f_S\left(x_S^{(i)}\right)\right) \tag{4.8}$$

where S is any subset of variables, F the entire set of variables, L is a given Loss function to be evaluated among two different f predictions. We are computing the Feature

Importance of the variable X_j for the unit $x^{(i)}$.

Algorithm 3 `Compute Shapley Values`

Require: f: black-box model; D: dataset on which to compute attributions, whose generic unit is $x^{(z)}$; L: loss function to compute the difference in prediction

for each variable X_j **do**
  for each combination of variables S (not including X_j) **do**
    for each observation $x^{(i)} \in D_{train}$ **do**
      generate new $\hat{D}^S$ dataset, with generic unit $\hat{x}^{(i)} = \left[x_S^{(i)}, x_{F\setminus S\cup j}^{(z)}\right]$
      generate new $\hat{D}^{S\cup j}$ dataset, with generic unit $\hat{x}^{(i)} = \left[x_{S\cup j}^{(i)}, x_{F\setminus S}^{(z)}\right]$
      predict $\hat{Y}_S = f(\hat{D}^S)$ and $\hat{Y}_{S\cup j} = f(\hat{D}^{S\cup j})$
    Compute the average $\Delta_j^{(S)} = \frac{1}{n}\sum_{z=1}^{n} L(\hat{y}_S^{(z)}, \hat{y}_{S\cup j}^{(z)})$
  return Compute weighted average Shapley$(X_j) = \sum_{S\subseteq F\setminus j} \frac{|S|!|(F-S-1)|!}{|S|!}\Delta_i^{(S)}$

Shapley values choose a single point $x^{(i)}$ and compute the average contribution of any group of variables using new fictitious points $\hat{x}^{(i)}$, created by assigning any possible value to the variables X_C not considered in the S coalition. This procedure is strongly reminiscent of the PDP marginalization, though applied on a single dataset unit. Shapley values are hence affected by the main PDP drawback -*creating irrealistic new points*-, however Shapley mitigates it by averaging the X_j contributions on any possible coalition.

Due to the combinatorial search in the variables' coalitions, exact Shapley Values have computational burden $\mathcal{O}(nd!)$, which means that the calculation becomes already unfeasible for arguably small d values (usually when $d > 8$).
Lundberg and Lee [78] propose an approximated solution through the KernelSHAP algorithm. Notably, KernelSHAP computes $\Delta_j^{(S)}$ on a restricted number of variables' combinations (by default 2048). The combinations are chosen to maximize the relative weights $\frac{|S|!|(F-S-1)|!}{|S|!}$, so as to obtain a more accurate approximation of the weighted average. To compute each $\Delta_j^{(S)}$, only 100 units $\hat{x}^{(z)}$ are employed, chosen at random.
Thanks to these tricks, KernelSHAP computes the Shapley values approximation SHAP(X_j) for a single datapoint $x^{(i)}$, in a reasonable time. The SHAP values for the specific unit $x^{(i)}$ are regarded as ***Local attributions***. SHAP ***Global attributions*** are easily obtained by averaging the Local SHAP values over the considered dataset, thanks to the additivity property.

From a Decomposition Rule point of view, SHAP(X_j) contains the main effect f_j as well as a portion of importance of any interaction effect $f_{j,\cdot}$. The contributions of the interactions are fairly split among the interacting variables, instead of being fully considered in each feature attribution, as in PDP. Another desirable property is that Local SHAP Values sum up to the unit prediction $f(x^{(i)})$ -*the baseline is required as well*-.
However, even if KernelSHAP displays an enormous computation speed-up compared to the original Shapley Values, the calculations are still quite slow. This makes KernelSHAP

useful to compute local SHAP Values, but usually not suited to achieve ***Global attributions***. Moreover, it is important to remember that SHAP values approximate the original Shapley Values, hence SHAP might contain some bias, especially when we sample a small number of units to compute $\Delta_j^{(S)}$ or few variables' combinations S.

A faster SHAP implementation, called TreeSHAP [77], is available only for Tree-based models. It enables Global Attributions in reasonable time, exploiting the specific Tree structure to compute Local SHAP values for the entire dataset in just one stroll down the Tree. We consider it as one of the first frameworks to provide ML models insights on localized regions of the model domain -*valid for a specific individual*-, and about the entire ML model surface at the same time. Unfortunately, this desirable property is restricted to Tree-based models only.

In general, feature attribution methods rely on a specific dataset to compute the importance of each variable, leading to the crucial question: which dataset to employ to produce the explanations?
We usually dispose of both D_{train} and D_{test} datasets when building a prediction model and both of them could be used in the explanation quest. The golden rule of prediction models is to split the database in an impartial way, obtaining D_{train} and D_{test} datasets belonging to the same data distribution, i.e. the DGP. When this is true, there should be no difference in using one over the other. Although i) D_{test} is usually smaller than D_{train}, and ii) sometimes the ML model might be overfitted, i.e. it learnt subtle patterns specific only to D_{train}. The first point is clearly in favour of using D_{train}, while the second suggests using D_{test} for explanation purposes. Moreover, if we require local explanations for a D_{test} (D_{train}) datapoint, it is necessary to use the dataset containing it, i.e. D_{test} (D_{train}). In conclusion, no answer can be given a priori on which dataset to use for feature attribution explanations. Depending on the specifics of the problem at hand, we might recognize that one of the issues above represents our situation, and we might opt to use the best dataset in this case.

4.2.0. Surrogate Models

A surrogate model is an interpretable model trained to match the predictions of a black-box model. The idea is not new, surrogate models have already been used in different situations where the exact model is too expensive to query or too complex to be used directly. A classic example is Bayesian Optimization, where we strive to minimize a complex function and we exploit a surrogate to evaluate its gradient and find the most promising direction to explore.

In the explainability setting, the surrogate model consists of the explanation itself, hence its only requirement is to be interpretable. As long as interpretability is preserved, the surrogate model can be used for either ***feature attributions*** and ***what-if scenarios***, i.e. understand how a given modification of the input will affect the black-box model pre-

diction. This feature is specific to surrogate models, since they learn an $g$mathematical function which emulates the ML f function. We can inspect the $g$hyper-surface to understand f behaviour when changing some inputs. Inspecting the surrogate model is usually faster and simpler than inspecting the ML model directly.

Although it is very important to build a reliable Surrogate Model, i.e. $g$should be very similar to f, otherwise any insight gained on $g$could not be translated to f. One way to measure how well the surrogate replicates the black-box model is the R-squared measure:

$$R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{\sum_{i=1}^{n} \left(g(x^{(i)}) - f(x^{(i)})\right)^2}{\sum_{i=1}^{n} \left(f(x^{(i)}) - \overline{f(X)}\right)^2}$$

where $\overline{f(X)}$ represents the average of f predictions on D_{train} , while SSE and SST respectively stand for Sum of Squared Errors and Sum of Squares Total.

Surrogate models can represent both global or local explanations, depending on the domain on which we train $g$to emulate f. Global Surrogates aim at approximating f on its entire domain $\mathcal{X}$, while Local Surrogates target a restricted region of the $\mathcal{X}$ domain and provide explanations valid only in that range.

The idea of approximating globally a complex model such as a deep neural network by a ***simpler surrogate model*** relying on a decision tree can be dated back at Craven and Shavlik [20]. The proposed method, TREPAN, approximates the outputs of the network on the training set by a decision tree, choosing the splits using the *gain ratio* criterion [98]. More recently, inspired by Gibbons et al. [32], Zhou and Hooker [127] proposed to use another split criterion, based upon an asymptotic expansion of the Gini criterion. This approach is model-agnostic since it does not rely anymore on the neural network structure, although it is still limited to the classification setting. However, since surrogate models must be interpretable, they do not usually have the same expressivity as a complex black-box model. Depending on the degree of accuracy requested for the Tree Global Surrogate, the model is frequently either too simple and not really resembling f or too complex, achieving a good f approximation but losing the interpretability requirement.

Concerning Local Surrogates instead, the approximation is usually more reliable since they focus on a small region. Under the right circumstances, the interpretable model is capable of faithfully approximating the black-box one. The Local Surrogate techniques usually differ for: i) the choice of the local region, ii) the interpretable model to employ. In the following, we describe some of the most well-known Local Surrogate explanation frameworks.

LIME

LIME [93] provides a number of explainable models which closely resemble the original model behaviour. Each model is specific for an input point $x^{(i)}$: only in its neighbourhood the explainable model's predictions are guaranteed to be very close to the black-box ones. This peculiarity places LIME among the Local Explainability tools. The technique can

be used on Images and Text Data, as well as on Tabular Data. Slight differences apply to be able to use the same algorithm on different data sources. Although, given the purpose of this thesis, we will focus on the Tabular domain only. LIME aims to approximate the black-box model f with a simple function $g$around the point of interest $x^{(ref)}$. $g$is required to lie in the class of explainable models.

$$\begin{aligned} f &: \mathbb{R}^d \to \mathbb{R}, \quad \text{black-box model} \\ g &: \mathbb{R}^p \to \mathbb{R}, \quad \text{explainable model} \end{aligned}$$

where d is the number of features employed by the black-box model, to make predictions about the response variable. The explainable model$g$uses only p of the original d variables, in order to reduce the complexity. Solving the following optimisation problem, we obtain the function $g$most similar to f in the neighbourhood of $x^{(ref)}$.

$$\begin{aligned} arg\min_{g} \; & L(f, g, \pi_x) + \Omega(g) \\ \Omega(g) &: \text{complexity of } g \\ L &: \text{loss function} \\ \pi_x &: \text{weight assigned according to } \hat{x}^{(i)} \text{ proximity} \end{aligned}$$

To this end, LIME relies on generating new points $\hat{x}^{(i)}$ on the entire feature space $\mathcal{X}$, and computes the relative $f(\hat{x}^{(i)})$ ML predictions. In order to account for locality, LIME assigns a specific weight π_{x_i} at each $\hat{x}^{(i)}$ point, based on its distance from $x^{(ref)}$. The new dataset $\left[\hat{X}, f(\hat{\hat{X}})\right]$ is employed to train an explainable model in a weighted fashion.

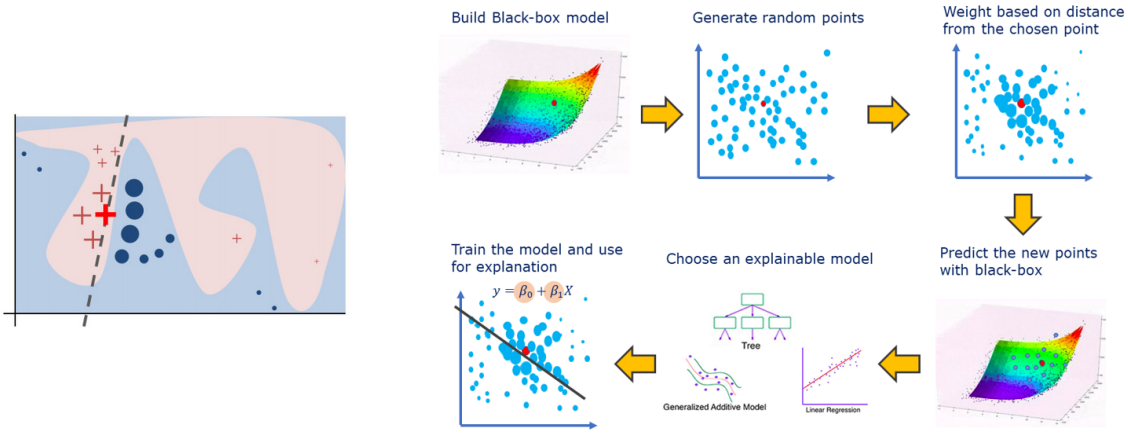


Figure 4.5: ***Left Panel***: LIME's modus operandi: the goal is to approximate the tangent to the ML model -*in this case a classification model separating red and blue regions*- in the neighbourhood of the red $x^{(ref)}$ point [93]. ***Right Panel***: LIME Algorithm Steps

By default, LIME uses a Linear model as explainable model, specifically Ridge Regression. Such choice enables LIME to find an ***approximation of the tangent plane*** to the ML

surface, in the $x^{(ref)}$ point we want to explain. Retrieving the tangent analytically is an unfeasible task, since we don't have a parametric formulation of the function f, besides the ML surface may have a huge number of discontinuity points preventing the existence of a proper derivative and tangent. LIME overcomes this by using a Ridge Linear Model to fit points on the ML surface, in the neighbourhood of the reference individual. In Figure 4.5, the Left Panel provide a visual intuition of LIME approximation to the ML tangent, while the Right Panel contains a schematic summarization of the algorithm's steps.

The explainable model is usually exploited to understand which variables are the most important for the ML prediction of the specific $x^{(ref)}$ individual, i.e. the higher the coefficient, the bigger the variation in the value of the response variable when the feature is changed. The sign of the coefficient tells us the direction of the variation, namely if we will face a decrease or an increase in the output value. It is important to stress that the LIME feature importance is valid only in the neighbourhood of the chosen individual, hence units with quite different attributes may have significantly different attributions. As an example, Figure 4.6 describes LIME feature importance for the Survival model built on NHANES dataset (Chapter 3.2) on a specific Female person aged 49. The Age coefficient shows us how ageing one year more, i.e. becoming 50, increases the death risk by 0.79 base points. This value would be different for a Male person with different age -*elder people would present higher age-associated death risk, since it increases superlinearly with age*-.

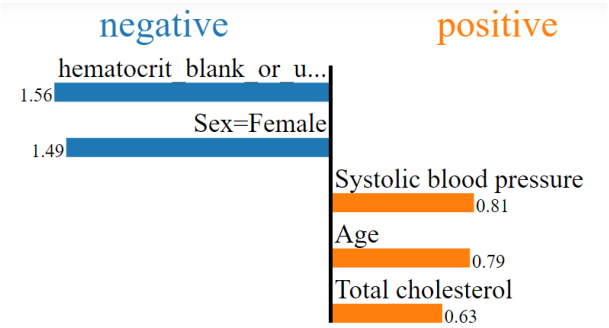


Figure 4.6: LIME helps understand and rank the major death risk factors for the specific individual of the NHANES dataset

Apart from feature importance, the LIME surrogate model can be also used to test what-if scenarios, such as: "If I were to earn 500$ more a year, how many points would I gain on my credit score? " Also for what-if scenarios, we can only test local (small) changes with respect to the original attributes of the chosen individual. Such a what-if tool is available only for surrogate models. It is not valid for explanations based on feature attributions, since they don't rely on prediction models.

LORE

Guidotti et al. [45] propose the LOcal Rule-based Explanations (LORE) framework, which consists of a tree local surrogate model, applicable to any ML model, but restricted to

the classification setting.
The concept is similar to LIME, but the interpretable surrogate and the generation step are different. In fact, they propose to employ genetic algorithms to generate a dataset of instances which are possibly very similar to our reference individual. At the same time, LORE requires the new dataset to contain a balanced amount of positive and negative examples x^+, x^-. The similarity of x^+, x^- towards $x^{(ref)}$ guarantees we are thoroughly exploring the classification boundary around $x^{(ref)}$, having examples on both sides of it.

The genetic algorithm allows us to keep the balance through the generations, while at each new step increases the similarity of new units with the reference. Hence the generation phase produces points that are already local and do not need any weighting step: each generated point x has the same importance π_x in the training phase of the surrogate model.
Compared to LIME, LORE requires more training time to allow the genetic algorithm to run, but at the same time guarantees a ***good local coverage*** of the manifold around the reference point. This makes LORE generation more efficient than the LIME one, producing only relevant points for the explanation model training.

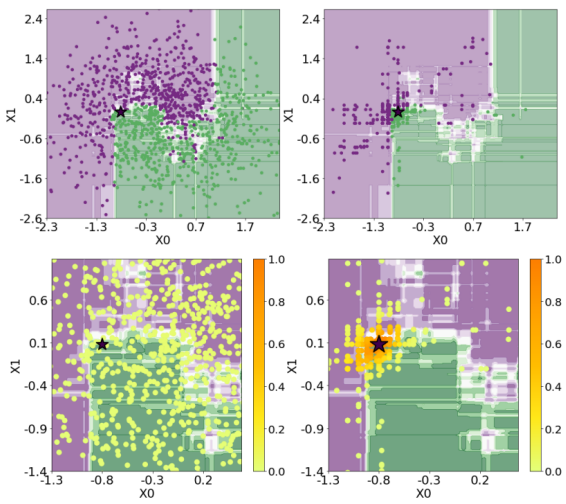


Figure 4.7: Comparison between uniformly random and genetically generated points, considering as reference the star point. ***Top*** Uniformly random (***left***) and genetic generation (***right***). ***Bottom*** Density of random (***left***) and genetic (***right***) generation. Courtesy of [45]

The explainable model itself is a Decision Tree instead of a model belonging to the Linear Regression family. The explanations would be in form of decision rules instead of variables' coefficients. As a byproduct of the surrogate decision tree, LORE provides also counterfactual explanations, i.e. units possibly the most similar to the reference, but with different class prediction. It does so by inspecting the trained Tree model and retrieving the closest rule to the one explaining the reference individual. By means of this rule, LORE creates the most similar unit to the reference, with a different prediction class, i.e.

the counterfactual.

ANCHORS

Ribeiro, Singh, and Guestrin [92] propose ANCHORS, a local explanation framework grounded on the same algorithm as LIME and LORE. The idea is to exploit Decision Rules, called Anchors. This means the framework approximates the f function using a Decision Tree as in LORE. As its predecessor, ANCHORS can be employed in a classification setting only.

The goal of the framework is to find local Anchors, which are possibly valid also on a global scale. Although the most important requirement is to obtain a reliable rule, for this reason the practitioner is required to set a minimum coverage level θ: the Anchors should yield the same prediction as the f model on at least a θ-percentage of the units complying with the rule. The framework hence minimizes the number of variables required to build the Anchors, under the θ constraint.

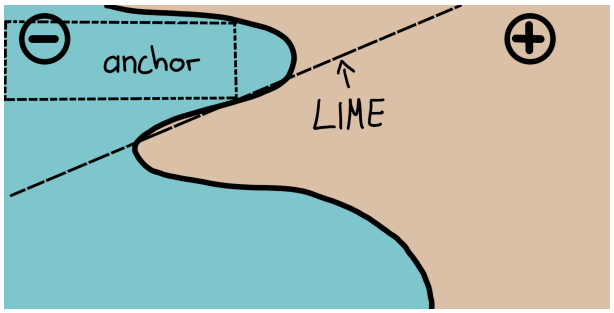


Figure 4.8: It describes well the main differences of using a Linear model or a Decision Tree as explainable models: the former gives an approximation of the f tangent describing how we expect the prediction to change when moving on the variables space, the latter instead finds a decision rule characterizing all the units belonging to the same patch as the $x^{(ref)}$ reference unit. Courtesy of [92].

Implicitly, ANCHORS tries to fit the largest neighbourhood size around the $x^{(ref)}$ point, guaranteeing to obtain an accurate explanation. In fact, considering fewer variables in the final decision rule corresponds to having more directions along which to generate the sample points, hence a larger neighbourhood. Differently from LIME and LORE, the ANCHORS neighbourhood is global for certain variables -*the ones not included in the decision rule, since the explanation is valid all over their domain*-, while restricted to local for the features considered in the rule.
Hence, we consider it as yet another local method, but its local explanations strive to be applicable as widely as possible. It can be considered an early attempt to push local explanations to be more global.

4.3.0. Evaluation of Explainability methods

Evaluating interpretability methods is quite challenging: for a given machine learning model, there is rarely a ground-truth available, telling us which features were used for a given prediction.

Zhou et al. [126] outline two main evaluation methodologies: ***human-centered***, and ***functionality-grounded*** quantitative *-note that the terminology was introduced by Doshi-Velez and Kim [22]-* evaluations. We focus on the latter, since human-centred evaluations are hard to obtain and can be unreliable. Among functionality-grounded evaluations, Zhou et al. [126] distinguish quite a number of ways to evaluate attribution-based explanations:

Monotonicity

Monotonicity is defined as the correlation between the absolute value of the attributions at a given point and the expected loss of the model restricted to the corresponding feature. Intuitively, this takes high values if the model is very imprecise when it does not know the feature at hand. This is Metric 2.3 in Nguyen and Martínez [88]. Following their notation, for point $x^{(i)}$ in the D_{train} dataset, we compute on the one hand a vector of attributions $\alpha \in \mathbb{R}^d$ and on the other hand a vector of expected loss $e \in \mathbb{R}^d$. Monotonicity is then defined as the Spearman rank correlation [107] between $|\alpha|$ and e. Formally, for all $j \in [d]$, e_j is defined as

$$e_j = \int_{a_j}^{b_j} \ell(f(x^{(i)}), f_j(x)) p(x) \mathrm{d}x \,, \tag{4.9}$$

where f_j is the restriction of f to coordinate j *-keeping all other coordinates equal to those of* $x^{(i)}$ -, p is a probability distribution on $[a_j, b_j]$ *-*a_j *and* b_j *represent the boundaries of* $\mathcal{X}$ *along dimension* j-, and ℓ is the squared loss. p is usually taken as the uniform distribution $\mathcal{U}(C)$ on a compact set C.
Based on the choice of C it is possible to define: ***local monotonicity*** if we restrict C to a local neighbourhood of the $\mathcal{X}_j$ feature space, or ***global monotonicity*** if we consider the entire feature space $[a_j, b_j]$. The two Monotonicity metrics check how well the attributions reflect the variations in prediction locally or on the whole input space along feature j.

Recall of Important Features

For certain models, we can actually be certain that some feature attributions should be 0. This is the case, for instance, if we force our model f to use only a subset of the features. As a measure of the quality of explanations, one can then compare the top features selected by an attribution method to the set of true features, known a priori. Let us call T the set of true features. For any given example $x^{(i)}$, we can define T^{x_i} the set of the top $|T|$ features, ranked by $|\alpha_j|$. Originally proposed by Ribeiro, Singh, and Guestrin [91], the recall of important features is then defined as

$$\rho(x^{(i)}) := \frac{|T \cap T^{x_i}|}{|T|} \,. \tag{4.10}$$

Intuitively, $\rho(x^{(i)})$ is large -*close to one*- if the considered attribution method gives large attributions to the true features.

Sensitivity

Two inputs $x^{(i)}$,$x^{(j)}$ with different predictions $f(x^{(i)}) \neq f(x^{(j)})$ and differing only by the value of a single feature X_j should receive a non-zero attribution α_j for this feature. This is called Sensitivity in Sundararajan, Taly, and Yan [109]. Depending on the distance between the two X_j values on $x^{(i)}$,$x^{(j)}$, we can define both local and global sensitivity -*the local version is described in Yeh et al. [119]*-.

Alternatively, one can consider the converse notion, ***non***-sensitivity Nguyen and Martínez [88, Metric 2.4]: considering the vector of attributions α and the vector of dummy variables `Irr` indicating the irrelevant features of the black-box f model, we count the concordance between $\alpha_j = 0 \implies \texttt{Irr} = 1$. In order to use this metric, the practitioner must know which features are irrelevant to f a priori.

Effective Complexity

The Monotonicity and non-sensitivity metrics consider only individual feature attributions, while the effective complexity takes into account feature interactions.

The idea is to retrieve the minimal set of important features -*ordered by the attribution scores*- such that the expected performance of the model restricted to non-important features is less than a given threshold Nguyen and Martínez [88, Metric 2.5]. Usually the threshold is quite small, to ensure that the non-important features have a negligible effect on f. Since the test is performed on groups of features together, their interactions are accounted for.

The size of the important features set is the Effective Complexity of the explanations. We aim for small values, i.e. simple and effective explanations. This metric evaluates explanations on a global scale, considering the features' effect on the entire model.

Remove and Retrain

The idea is very similar to Perturbation Importance: we consider replacing the most important features, i.e. corresponding to highest attribution scores α_j, by non-informative values. They might as well be a permutation of the original values. The model f is retrained and we compute the drop in performance.
The metric was originally proposed by Hooker et al. [58, ROAR], where it is shown that many methods do not perform better than random. A big drawback is the high computation cost, since we need to retrain the black-box f.

Selectivity

The basic idea is to evaluate how well global attributions rank the features in order of importance. To do so, we remove one feature at a time -*either replacing it with its permutation*

or retraining the f model-, in order of importance given by the attributions, and compute the drop in the model performance.
It is possible to compute the Area Under the ROC Curve (AUC) of the performance results at each stage, obtaining a quantitative value. Montavon, Samek, and Müller [84] gives credits to Bach et al. [3] and Samek et al. [99].

While the metric takes values in $[0, 1]$, there is actually an upper bound on the maximum value that can be achieved. The upper bound depends on f, i.e. black-boxes in which some features have similar high importance have inherent Selectivity upper bound lower than models with only one or few features highly relevant.

Mutual Information

Mutual Information MI measures the non-linear dependence between two variables. Nguyen and Martínez [88, Metric 2.1] describe how it can be used to assess the fidelity of the explanations towards the f model, by computing $MI(f(X), \alpha)$, where α is the matrix of feature attributions computed on a given dataset X for any point $x^{(i)}$, while $f(X)$ is the vector of f predictions on X. The higher the dependence, the more reliable the explanation method is in mimicking f.

Chapter 5

LIME in Detail

As explained above, Surrogate models provide the user with both ***feature attributions*** and the possibility of running ***what-if scenarios***. The two-fold usage makes the surrogate model class our preferred choice.
Although, we recognize that the current Global surrogate techniques are either too simple and not really resembling f or too complex, hindering a clear and simple interpretation of the model. On the contrary, Local surrogate models approximate only a small portion of the f surface, making them simple yet possibly very close to f. Local surrogates have already been successfully employed in many different contexts, such as on Intensive Care data [65], cancer data [123],[86] or Credit Scoring domain [117].

In this chapter we are going to focus specifically on LIME, since it represents the first algorithm proposed to build local surrogate models for explanation purposes. More recent local surrogate techniques usually consist of a refined version of the vanilla LIME algorithm, in which some steps are modified or improved. As such, LIME deserves specific attention to understand the pros and cons related to each step of the algorithm.

5.1.0. Generation Step

LIME relies on generating new points instead of using the original dataset, to ensure good coverage of the domain space $\mathcal{X}$.

The original dataset is still employed to compute relevant statistics about the variables, i.e. univariate means μ_j and variances σ_j^2 for each feature X_j. Although, sometimes privacy and secrecy standards require to not disclose the original data. It can also be the case that the practitioner has access to the trained ML model only, but no access to the D_{train} data. Interestingly, LIME does not strictly require the original dataset as input, as long as the quantities of interest are provided. *Notice that this is not the case for most of the feature attribution methods, which instead necessitate* D_{train} -. LIME requires indeed very little information about the original data, in an aggregated form only, making it suitable for sensitive situations.

The new units $\hat{x}^{(i)}$ are generated from a multivariate normal distribution $\mathcal{N}_d(\mu, \Sigma)$,

where $\mu = [\mu_1, \cdots, \mu_d]^T$, while Σ is the covariance matrix with generic diagonal element $\sigma_{j,j} = \sigma_j^2$, off-diagonal elements $\sigma_{z,j} = 0$ when $z \neq j$. Given the above Σ matrix, the LIME generation step assumes that the generic variables' pair X_j ,X_z are independent. Such assumption produces ***True to the Model*** explanations, since it generates $\hat{x}^{(i)}$ units also in low or null probability areas of $\mathcal{X}$.
From a space exploration point of view, we consider the Normal distribution as probably not the best choice, since it produces more samples in the central part of the data manifold (around the μ vector), while it neglects the far-away regions. When $x^{(ref)}$ has values quite different to μ the generation produces many $\hat{x}^{(i)}$ sitting far-away from $x^{(ref)}$, hence not particularly useful.
An interesting alternative is to use a Multivariate Normal Distribution where the μ vector corresponds to the $x^{(ref)}$ unit values, while Σ is the same as above. This induces a much more specific generation around the reference point, although it might produce points sitting out of the variables boundaries of the original dataset.

LIME generation phase is global, i.e. the n points $\hat{x}^{(i)}$ are generated all over the $\mathcal{X}$ domain. The weighting step subsequently takes care of enforcing locality by assigning a different weight to each $\hat{x}^{(i)}$. In the generation step, we obtain an array of values $\hat{X}$, with univariate distributions similar to the original dataset, although we don't have any clue about the Y values corresponding to $\hat{X}$. LIME computes the $\hat{y}$ values as ML predictions $f(\hat{x})$, making the prediction function f a fundamental LIME input.

Notice that LIME properly ***handles both classification*** and ***regression*** models. In the regression setting, for any input $\hat{x}^{(i)} \in \mathcal{X}$, $f(\hat{x}^{(i)})$ is simply a real-valued quantity of interest, whereas in the classification case $f(\hat{x}^{(i)})$ corresponds to the pseudo-probability of $\hat{x}^{(i)}$ falling into a given Y class, eg. $f(\hat{x}^{(i)}) = P(\hat{y}^{(i)} = 1|\hat{x}^{(i)})$ if the class to be explained is class 1. Using prediction probabilities makes the $\hat{Y}$ data continuous (specifically in the range $[0, 1]$), which allows LIME to train an explainable Linear Regression model on them. This is a recurrent caveat in explanation methods, in order to extend the techniques to both regression and classification models, without modifying the underlying explanation algorithm.

After computing f predictions, the $\hat{X}$ dataset undergoes a rescaling process: each X_j feature is standardized by subtracting μ_j and dividing by σ_j. The response variable $\hat{Y}$ is left untouched. Standardization pushes each variable on the same scale (which is approximately $[-5\sigma_j, +5\sigma_j]$ according to Chebishev inequality), which allows us to directly compare feature attributions of the explainable model.

Why do we need the sampling step?

Zafar and Khan [120] propose to get rid of the sampling step, by using only the original dataset points. The result is the Deterministic LIME (DLIME) approach. Their choice basically converts the LIME framework in a ***True to the data*** approach, since the local surrogate is computed exploiting points stemming from the true data distribution. Two remarks to this method: i) it is mandatory to provide the original dataset to DLIME, losing the privacy perks of the original LIME framework, ii) the original dataset does not

guarantee an appropriate coverage of the $\mathcal{X}$ space.
The latter point is especially relevant, since the local explanation framework requires to consider small portions of $\mathcal{X}$, which frequently do not contain enough training points to build a reliable and significant local surrogate. This happens especially when we consider points close to data manifold borders, as illustrated in Figure 5.1. In fact, considering a small neighbourhood (dark green area) DLIME would fail since we do not have enough points to train Linear Regression; a larger neighbourhood (light green) includes quite distant points which prevent a good local approximation.

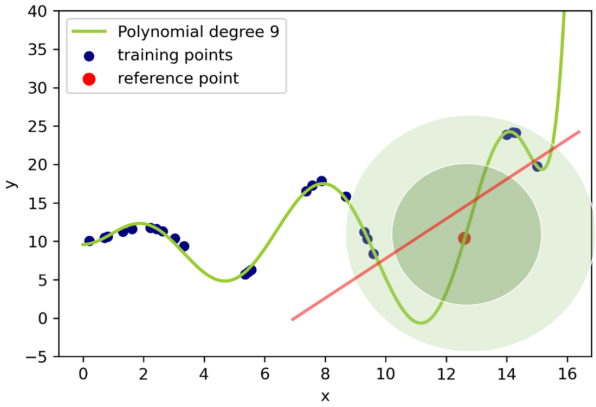


Figure 5.1: We consider the same Toy Dataset and Polynomial ML model in Chapter 3.4.2, expanding the X domain upwards. We illustrate how LIME explanations trained only on D_{train} data can be very shaky in sparsely populated regions.

Through this visual and non-rigorous example, we want to highlight how the generation step is extremely important to ensure good coverage of the f surface.

Global vs Local Generation

Why LIME generates points over the entire domain $\mathcal{X}$ instead of focusing on local generation in the $x^{(ref)}$ neighborhood?

The concept of local generation is appealing and brings various advantages, among others a better exploitation of the generated points and a reduced bias risk in the local explainable models caused by non-relevant units.
However, local generation requires to choose the neighbourhood size in which to produce new points and there is still no consensus on how to select the best size. It is important to remark that the proper size cannot be fixed a priori to a given range for each different $x^{(ref)}$ point, in fact it is point dependent. Consider to rely on Linear Models as local LIME surrogates, the neighbourhood should include all the quasi-linear area of the ML curve around the reference point, therefore it depends on the local curvature of $f(x)$. Hence, different points have different proper sizes for the linear local region, as shown in Figure 5.2.
Local generation of the $\hat{x}^{(i)}$ samples is considered an active research area, and few different papers explored this concept.

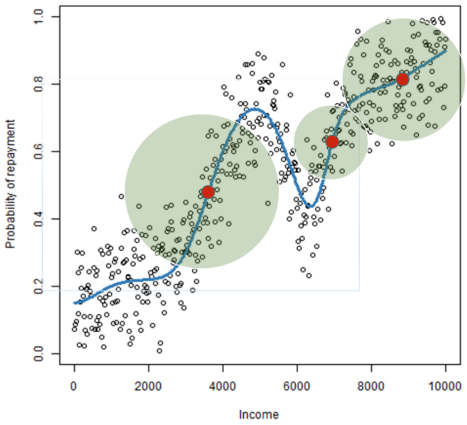


Figure 5.2: The best neighbourhood size depends on the reference point and the curvature of the ML function around it.

In particular LORE [45] (See Chapter 4.2) proposes a genetic algorithm guided generation to produce points close to $x^{(ref)}$ and at the same time interesting from a classification point of view, i.e. on both sides of the decision boundary. LORE implicitly inspects the decision boundary and provides points $\hat{x}^{(i)}$ that are meaningful to describe it. Although, it does not provide a clear formulation of the local neighbourhood, making it difficult to understand how far we can push the ***what-if scenarios*** (we don't exactly know if a given change in variables' values produces a what-if point still lying in the $x^{(ref)}$ neighbourhood). It is important to remember that what-if scenarios are valid only when the variables changes are included in the local neighbourhood. This is a major issue of LIME as well.

Also Laugel et al. [73] consider the classification setting only, suggesting to grow hyper-spheres of increasing l_2 radius until the surface of the spheres touches the decision boundary. This procedure guarantees to find the boundary point $x^{(bound)}$ closest to $x^{(ref)}$. The local generation is carried out by means of the hyper-spere of radius equal to the l_2-distance between $x^{(bound)}$ and $x^{(ref)}$, sampling from a Uniform distribution in this region. Thanks to the growing hyper-spheres, we know exactly the boundaries of the Local Surrogate model. Although the radius choice as l_2-distance between $x^{(bound)}$ and $x^{(ref)}$ seems somewhat arbitrary, while we argue that the size of the neighbourhood should depend on the degree of non-linearity that f exhibits in the local region -*the aim should be to consider a part of the space in which f can be considered almost linear*-.
Moreover, both techniques rely on the decision boundary of classification models, preventing them to be used in regression settings.

On the other hand, vanilla LIME employs global generation paired with a weighting step to assign reduced importance to distant $\hat{x}^{(i)}$ units. In the following, we detail how the local weighting is carried out, along with its pros and cons.

5.2.0. Weighting Step

Locality is enforced through a kernel function, the default is the RBF Kernel (Formula 5.1). It is applied to each point $\hat{x}^{(i)}$ generated in the sampling step, obtaining an individual weight. The formulation provides smooth weights in the range $[0, 1]$ and flexibility through the kernel width parameter kw.

$$\pi_{x_i} = \exp\left(-\frac{||x^{(i)} - x^{(ref)}||^2}{kw}\right) \tag{5.1}$$

The RBF flexibility makes it suitable to each situation, although it requires proper tuning: setting a high kw value will result in considering a neighbourhood of large dimension, shrinking kw we shrink the width of the neighbourhood.

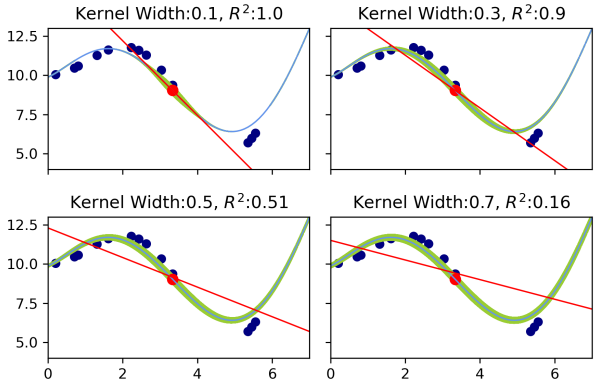


Figure 5.3: LIME explanations for different kernel widths. Notice how too large kw distort the local linear model -*testified by the* R^2 *measure as well*-

In Figure 5.3, LIME generated points are displayed as green dots and the corresponding LIME explanations (red lines) are shown. The points are scattered all over the ML function, with size proportional to the π_{x_i} weight assigned by the RBF kernel. Small kernel widths assign significant weights only to the closest points, making the further ones almost invisible. In this way, they do not contribute to the local linear model.
The concept of locality is crucial to LIME: a too large neighbourhood may cause the LIME model not to be adherent to the ML function in the considered neighbourhood. This makes the kernel width kw the most important parameter of the entire LIME framework.

5.3.0. Feature Selection

In order to obtain a human-understandable linear model it is mandatory to use only a bunch of features: explanations including the whole set of features may turn out to be messy and confusing. Therefore LIME performs also a feature selection step, to present the user with the most important variables only.

Different choices are available: we can use feature selection techniques already built into the model training, such as Lasso or Stepwise Regression, or we may train the linear model on the entire set of features and keep only the ones with the highest coefficients.

In principle, highest coefficients should be preferred (because there is no noise in the $\hat{X}$ dataset), but using all the variables can cause collinearity issues in the LIME linear model. In fact, linear models usually do not cope well with high-dimensional datasets, since the design matrix might be ill-conditioned, causing unstable estimates of the best linear model.
Lasso Regularization usually takes care of such situations, at the price of introducing a small bias in the estimation. Throughout the entire thesis we employ Lasso regularization, considering that its benefits firmly overcome the slight bias introduced.
In the Python LIME implementation, the number of variables to be retained, namely p, is chosen by the practitioner.

5.4.0. Local Model Step

On the standardised p-dimensional dataset, LIME applies an explainable model. The algorithm allows for any model to be used, although the default is Ridge Regression, i.e. a Linear Regression combined with a penalty related to the $\ell 2$ norm of the coefficients [56], used to prevent overfitting. The model training is done in a weighted fashion: each generated point contributes to the model according to its weight π_x.

Using Linear Models as local surrogates allows LIME to stick with the concept of approximating the f tangent in the local neighbourhood around $x^{(ref)}$. This viewpoint makes LIME explanations quite intuitive and appealing.

About Ridge Regression

We now assume a linear DGP (recall the definition in Chapter 2.1) of the form $DGP(X) = \alpha + \sum_{j=1}^{d} \beta_j X_j + \mathcal{E}$, containing some noise $\mathcal{E}$.
Ridge Regression [56] is simply a linear model: $\mathbf{E}(Y) = \alpha + \sum_{j=1}^{d} \hat{\beta}_j X_j$, which optimizes the estimates $\hat{\beta}_j$ to be similar to the real β_j DGP coefficients. The optimization is usually done through consecutive iterations using the Gradient Descent technique, but the model can also be solved in closed form, i.e. the exact solution can be computed in a single step. The main difference, compared to simple linear regression, is the penalty term used in the $\hat{\beta}$ estimation, which is proportional to the $\ell 2$ norm of the estimated coefficients $\hat{\beta}$ and is governed by the λ hyper-parameter:

$$\hat{\beta}_R = \left(X^\top X + \lambda I\right)^{-1} X^\top Y$$

where $\hat{\beta}_R$ stands for the Ridge estimated coefficients vector.

This technique is useful when dealing with noisy datasets (where the stochastic component $\mathcal{E}$ exhibits high variance σ^2) [115]. In fact, the presence of strong noise makes various sets of coefficients as viable solutions. Instead, tuning λ to its proper value allows Ridge to retrieve a unique solution.

In the LIME setting, the ML function acts as the DGP, while the sampled $\hat{x}^{(i)}$ points are the reference dataset. Recalling that the $\hat{y}^{(i)}$ coordinate of each point is given by ML

prediction $f(\hat{x}^{(i)})$, it is guaranteed they lie exactly on the ML surface by construction. Hence, no noise $\mathcal{E}$ is present in our $\hat{X}$ dataset. For this reason, we argue that the Ridge penalty is not needed, on the contrary it can be harmful and distort the right estimates of the parameters, as shown in Figure 5.4.

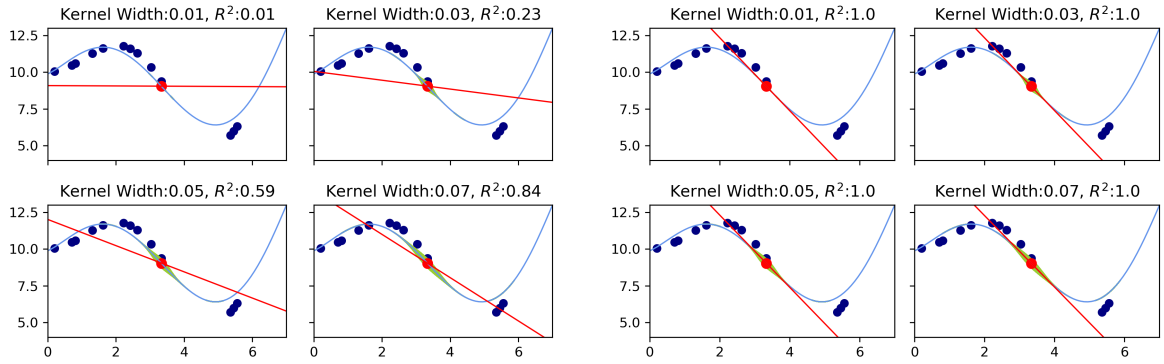


Figure 5.4: In the ***Right Panel*** Ridge penalty $\lambda = 1$ (LIME default) is employed, whereas in the ***Left Panel*** no penalty ($\lambda = 0$) is imposed. It is possible to see how the estimation gets severely distorted by the penalty, proven also by the R^2 values. This happens especially for small kernel width values, since each unit has a very small weight and the weighted residuals are almost irrelevant in the Ridge loss, which is dominated by the penalty term. To minimize the penalty term the coefficients are shrunk towards 0.

We demonstrated above that Ridge Regression does not bring any estimate improvement over simple Linear Regression in the LIME context, on the contrary, depending on how small the neighbourhood size is, we may incur strong ***distortion and bias***. Therefore, we suggest applying the LIME framework using simple Linear Regression, without any regularization term.

5.5.0. LIME Issues

We are going to mention some drawbacks and pitfalls LIME explanations might have, depending on the datasets they were trained on, the number of variables or the choice of hyper-parameters.

LIME fails in high-dimensional datasets

LIME is sensitive to the dataset dimensionality: when it is employed to interpret a Machine Learning model built using a huge number of variables, the local explanation is unable to discriminate between relevant and irrelevant features.

This phenomenon is due to the weighting kernel. Generally speaking, it can be considered as a similarity (or distance) function, thus it inherits the drawbacks of this class. As thoroughly described by Beyer et al. [5], in high dimensional datasets, chosen a fixed point, the distance to its nearest data point approaches the distance to the farthest one, as dimensionality increases. In simple words, with too many variables in the dataset, all the $\hat{x}^{(i)}$ points created in the generation phase have a similar distance from $x^{(ref)}$ and all of them are quite far from it, because euclidean distance metrics get rapidly too large in high-dimensional settings.
LIME applies the RBF kernel function to assign weights π_{x_i} before variable reduction, in fact we have no prior notion of important/unimportant variables -*hence we cannot compute RBF distance on the set of relevant variables only*-. Therefore, the kernel function considers all $\hat{x}^{(i)}$ as quite far and approximately at the same distance from $x^{(ref)}$. The explainable Linear model has almost no relevant points to rely on -*all points have a small weight*- resulting in a loss of the locality concept and consequently in a bad performance of the algorithm.

Such occurrence is shown in Figure 5.5: we used the Credit Scoring Dataset of Chapter 3.1 to train a Gradient Boosting model without any prior feature selection step, i.e. keeping all the 100 different features. GBM performance did not suffer from high-dimensionality, thanks to its ability to select relevant variables to be used at each step, although its LIME explanations are highly sensitive to a huge amount of input variables. As consequence, LIME has not been able to discriminate between important and irrelevant regressors. In particular, many features exhibit low values and almost all of them are equally important.

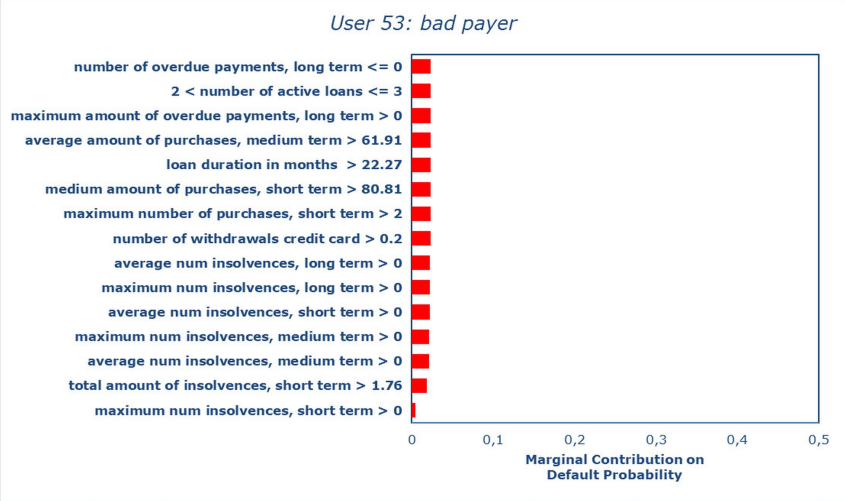


Figure 5.5: LIME explanations are not informative when applied to Machine Learning models with many input variables, in this case a Gradient Boosting model using 100 features, on the Credit Scoring Dataset of Chapter 3.1.

This weakness curbs LIME's employment on black-box models handling high-dimensional datasets. To date, it is a practitioner's duty to ensure the dataset dimensionality is low

enough for LIME to work well. This usually requires feature selection, upstream of data modelling.

LIME Stability issue

Consider choosing a specific individual and performing LIME on it, several times. Indeed, it is desirable to obtain the same explanations from each call.
Every time LIME is employed, it generates new data points, which follow the same distribution (law) but are different among distinct applications. This is due to the random nature of the sampling. Using different points it may happen to obtain divergent explainable models g, thus different explanations, for the chosen individual.
Based on this evidence, we define the concept of LIME stability: explanations derived from repeated LIME calls, under the same conditions, are considered stable when statistically equal. In [2] the authors provide insight about LIME's lack of robustness, a similar notion to the above-mentioned stability. Analogous findings also in [36]. On a more general perspective, [43] and [42]

Instability of the explanations is a major issue: obtaining substantially different LIME coefficients at each call prevents a full comprehension of the ML model, since we do not know which explanation to rely on. Moreover, instability is rarely taken into account in business projects, quite the opposite practitioners are often not aware of such a drawback and they rely on a single LIME call, considering the explanation as reliable without further checks. Instability awareness and checks to ensure explanations stability is the only way to provide useful and high quality explanations.

5.6.0. Improvements over vanilla LIME

Given the aforementioned issues intrinsic to the LIME method, a number of works addressed these concerns by proposing improved techniques. In general, the methods presented in the following share the main idea and programmatic step of the vanilla LIME algorithm, but change some of the steps to obtain better explanations.

Shankaranarayana and Runje [102] propose to additionally train a Denoising Autoencoder to reconstruct slightly corrupted training data -*corruption made by means of adding white Gaussian noise*-. The Autoencoder is then employed to weight the $\hat{x}_i$ instances generated by LIME: instead of using RBF kernel distance on the original fatures, ALIME computes weights on the Autoencoder latent space distances. Since the latent space dimensions are user defined, the number of features on the latent space can be kept low to avoid the curse of dimensionality when computing distances. ALIME allows to compute explanations on high-dimensional models as well, as long as we can guarantee that the Autoencoder has been properly trained.

Moreover, ALIME proposes to generate a unique sample $\hat{X}$ of new datapoints, to be used on each local LIME instance, weighting the datapoints differently based on the distance on

the given reference $x^{(ref)}$. Such approach indeed guarantees stability of the explanations, but may compromise their fidelity if we cannot ensure that enough generated datapoints are present in the $x^{(ref)}$ neighbourhood.

Bramhall et al. [8] suggest to use more complex approximations of the black-box model instead of simple degree 1 polynomials, in particular they suggest to use polynomials of degree 2 -*quadratic functions*-. On one hand, QLIME improves the degree of fidelity of the explanation since the approximation is more flexible and capable of adapting more on ML surfaces with non-linear curvature, on the other hand we loose the interpretation simplicity. Explanation coefficients are difficult to interpret in a what-if fashion: compute expected output of simple input modifications by using quadratic form coefficients is somewhat involved. Hence, QLIME looses the surrogate models perks where its coefficients can be used only as feature attributions.

Zhao et al. [125] propose to produce explanations as a weighted combination of prior knowledge on the local β parameters of the Linear approximation and the $\hat{\beta}$ coefficients obtained from LIME, framing the problem as Bayesian Inference.

Regarding the prior knowledge, the authors propose different directions among which: i) run LIME on similar instances as the reference datapoint and use the average $\hat{\beta}$ coefficients, ii) initialize β as the attributions of different explanation methods -*such as SHAP, GradCAM etc.*- on the same reference instance.

It is important to notice that prior knowledge of the β is assumed to be without uncertainty, hence reducing the instability of BayLIME predictions. The real challenge in this setup is to properly tune the weights given to the prior and LIME coefficients, governed by the λ parameter.

The framework shows good results in terms of robustness and fidelity to the underlying ML model behaviour, but may require a fair amount of computation. Moreover, other explainability methods are not immune to instability, while BayLIME considers their estimate as completely reliable when plugged in the weighted sum.

Chapter 6

LIME Stability Indices

Few papers already dealt with the instability issues described above, as their efforts can be grouped into two different approaches: i) get rid of the sampling step, ii) evaluate the post-hoc stability of the retrieved explanations.
The former idea is put forward by Zafar and Khan [120] (as already explained in Chapter 5.1), proposing to bypass the sampling step by using the training units only, and a combination of Hierarchical Clustering and K-Nearest Neighbour techniques. Although this method achieves stability, it may find a bad approximation of the ML function, in regions with only few training points.

On the other hand, the concept of evaluating the stability of already trained explanations, in a post-hoc fashion, has been tackled in different papers. The shared idea is to repeat the LIME method at the same conditions, and test whether the results are equivalent. From a theoretical point of view, Cosine Similarity could be an easy way to compare the explanation vectors retrieved by repeated LIME calls. Unfortunately, Cosine Similarity can be used only when vectors share the same coordinate system: LIME explanations retrieve possibly different important variables, making them not directly comparable. Among the various propositions on how to conduct the test, in [102] the authors compare the standard deviations of the Ridge coefficients, whereas [83] examines the stability of the feature selection step -*whether the selected variables are the same*-.

Although we consider recent work on the topic headed in the right direction, we feel more work has to be done in order to provide solid grounds and mathematical rigour to the metrics evaluating LIME stability.

For this reason, we frame LIME stability in a more rigorous way, to be subsequently able to formulate a new proposal for post-hoc stability evaluation.
Recall that LIME builds a linear local surrogate model g, using only a handful of variables $p < d$. Hence, g can be viewed as a mapping function between the set of variables and the respective coefficients:

$$g : \mathcal{F} \to \mathbb{R} \tag{6.1}$$

where $\mathcal{F}$ is the set of variables, of cardinality d. p out of d features will be associated with a value different from 0, the others $d - p$ variables will have 0 coefficient, meaning

they are irrelevant to the model. The formulation $g(\texttt{feat})$ indicates the coefficient value of the feature named `feat`, in the model g.

We perform m different calls to LIME on the model f and the individual x_i, obtaining m different explainable models $g_1...g_m$.

We want to: (i) check whether different g are composed of the same variables, (ii) compare the coefficients of the same variable among $g_1...g_m$ and test whether they can be considered equal.

To this purpose, we devise two complementary indices: the Variables Stability Index (VSI) and Coefficients Stability Index (CSI). The indices are designed for LIME frameworks using Ridge Regression as local linear models, in order to broaden their possible employment. In fact, simple Linear Regression can be considered as a special case of Ridge Regression, obtained by setting $\lambda = 0$. Deriving VSI and CSI on the Ridge Regression allows to use them on default LIME -*unaware practitioners using vanilla implementation can take advantage of the indices*-, although we still recommend using simple Linear Regression over Ridge Regression.

6.1.0. Variables Stability Index: VSI

The Variables Stability Index (VSI), whose steps are explained in Algorithm 4, addresses the first point, namely it compares the variables composition of the $g_1...g_m$ models.

We consider the set $C_2^m(g_1...g_m)$ of all possible combinations of the m explainable models, two by two. The generic element of $C_2^m(g_1...g_m)$ is the pair (g_α, g_β).

We define a measure of concordance among the two explainable models in each pair:

$$\begin{aligned} &\texttt{pair} = (g_\alpha, g_\beta) \\ &\mathcal{F}_\alpha = \{\texttt{feat} \in \mathcal{F} : g_\alpha(\texttt{feat}) \neq 0\} \\ &\mathcal{F}_\beta = \{\texttt{feat} \in \mathcal{F} : g_\beta(\texttt{feat}) \neq 0\} \\ &\textsc{concordance}(\texttt{pair}) = |\mathcal{F}_\alpha \cap \mathcal{F}_\beta| \end{aligned} \tag{6.2}$$

where $\mathcal{F}_\alpha$ and $\mathcal{F}_\beta$ represent respectively the variables used in the explainable models g_α and g_β. The CONCORDANCE function returns an integer value, namely the cardinality of the intersection between $\mathcal{F}_\alpha$ and $\mathcal{F}_\beta$, ranging from 0 to p. It represents the number of variables used by both g_α and g_β.

We evaluate the CONCORDANCE over all the pairs in $C_2^m(g_1...g_m)$ and we average them, obtaining the VSI index, ranging from 0 to 1. We express the index as a percentage: it now spans from 0 to 100, the more it approaches 100 the more the variables found in different applications are the same.

Algorithm 4 Variables Stability Index (VSI)

Require: $g_1 ... g_m$

1: $n = 0$

2: **for** pair in $C_2^m(g_1 ... g_m)$ **do**

3: $\quad n = n + \frac{\text{CONCORDANCE(pair)}}{p}$

4: VSI $= \frac{n}{\left|C_2^m(g_1 ... g_m)\right|}$ **return** VSI

6.2.0. Coefficients Stability Index: CSI

The equality between coefficients of the different $g_1 ... g_m$ models is now under investigation. In the following, we derive the statistical distribution of the coefficients and we rely on it, to create confidence intervals and possibly statistical tests.

It is a well-known result [37], that under the classic assumptions of Linear Regression, the coefficients are guaranteed to follow a Gaussian distribution. This is not sufficient, since we deal with Weighted Ridge Regression.

In [114], the distribution of the Ridge Regression estimator is given by the formula:

$$\hat{\beta}(\lambda) \sim \mathcal{N}\Big((X^T X + \lambda I_p)^{-1} X^T X \beta, \sigma^2 (X^T X + \lambda I_p)^{-1} X^T X [(X^T X + \lambda I_p)^{-1}]^T\Big) \quad (6.3)$$

where X is the matrix of observations. In our setting, X is composed of the points randomly sampled inside LIME. The matrix I_p stands for the identity matrix (dimensions $p \times p$). σ^2 is the variance of the $\mathcal{E}_i$ random variables describing the errors per each sampled point. Under the Regression assumptions the errors $\mathcal{E}_i$ are independent and identically distributed (IID) following a Gaussian law: $\mathcal{E}_i \sim \mathcal{N}(0, \sigma^2)$. λ stands for the Ridge regularisation coefficient. The vector β represents the true values of the coefficients in population, whereas $\hat{\beta}$ consists of the estimates of the true values, using the X dataset.

In our setting, we may consider the β values as the unknown coefficients of the best linear approximation of f in the neighbourhood of $x^{(ref)}$. LIME aims to provide $\hat{\beta}$ closest as much as possible to the unknown β values.

Concerning Weighted Regression, it is usually estimated via Generalised Least Squares (GLS) which guarantee the distribution of its estimators to be the following [62] :

$$\hat{\beta} \sim \mathcal{N}\Big((X^T W X)^{-1} X^T W X \beta, \sigma^2 (X^T W X)^{-1}\Big) \quad (6.4)$$

In the formula, W is the $n \times n$ diagonal matrix of weights per each unit. In our setting, the W matrix is populated by the kernel weights calculated on the distance of each sampled point from $x^{(ref)}$.

It is important to recall that σ^2 is an unknown value and we are requested to obtain an unbiased estimator inferred from the data. Such estimator takes the form:

$$\hat{\sigma}^2 = \frac{\mathcal{E} W \mathcal{E}^T}{n - p}$$

for the Weighted Regression, as stated in Johnston and DiNardo [62]. $\mathcal{E}$ stands for the vector of the errors per each sampled point: $\mathcal{E} = (\mathcal{E}_1 ... \mathcal{E}_n)$. As far as Ridge Regression is concerned, the variance estimator remains unchanged from the Linear Regression's one [114].

Using the building blocks stated before, we derive the distribution of the Weighted Ridge Regression estimator. Starting from the Ridge Regression law (Equation 6.3), we know [6] that the Gaussian distribution is invariant whenever we employ a matrix of known weights. This guarantee the Weighted Ridge law of the coefficients to be Gaussian. Its distribution is :

$$\begin{aligned}\hat{\beta}(\lambda) \sim \mathcal{N}\Big((X^T W X + \lambda I_p)^{-1} X^T W X \beta, \\ \sigma^2 (X^T W X + \lambda I_p)^{-1} X^T W X [(X^T W X + \lambda I_p)^{-1}]^T\Big)\end{aligned} \tag{6.5}$$

We provide also the formula for the variance estimator of the Weighted Ridge Regression [1]:

$$\hat{\sigma}^2 = \frac{\mathcal{E} W \mathcal{E}^T}{n - p} \tag{6.6}$$

where n is the number of data points sampled inside LIME, p denotes the number of variables considered in the explainable model.

Knowing the distribution of the coefficients, we might derive a test statistic to assess a null hypothesis of equality. This comparison can be carried out also among coefficients of two different regression models, as long as they were estimated on two independent samples drawn from the same law, as derived by Brame et al. [7] for the coefficients of two distinct Linear Regressions.

This assumption holds true in our experimental design, since the data are sampled from the features' distribution inferred from the original data. It means that the true generating distribution of $X_1 .. X_m$, i.e. the datasets sampled in repeated LIME calls, is identical, while the differences among them are attributable only to the sampling variance.
Unfortunately, the simplifications carried out in Brame et al. [7] and Greene [37] in order to derive the t-test statistic, rely on the equality of the expected value of the two coefficients taken into consideration. This is true in Linear Regression, but the framework breaks down using a regularisation technique such as Ridge: the regulariser trades off the unbiasedness of the estimator in exchange for a possibly strong reduction of the variance.

Since the estimator is not unbiased any more, the expected value is now depending on the design matrix X. As stated before, different LIME calls give rise to different design matrices, this implies the expected values of a specific variable, taken from two different

[1] In this formulation, we consider $\mathcal{E}$, as the errors of the Linear Regression model. In other words, the Weighted Ridge $\hat{\sigma}^2$ estimator is the same of Weighted Regression.
We may not use the errors of any Ridge model to calculate an unbiased estimator of the error variance σ^2, because the Ridge regularisation term decreases the variance. Using such errors would cause the estimator to be biased towards 0.

explainable models g_α and g_β, to be different: $\mathrm{E}[g_\alpha(\texttt{feat}) - g_\beta(\texttt{feat})] \neq 0$. This result causes the derivation of the t-test statistic to break down.

Testing the null hypothesis of equality has proven tricky and not easily solvable, hence we rely on the Gaussian distribution of the coefficients to construct 95% confidence intervals. To do that, we design the function ConfInt, taking as input a g's coefficient and giving back its confidence interval:

$$\begin{aligned}\textsc{ConfInt}(g(\texttt{feat})) = [&g(\texttt{feat}) - 1.96\ \sqrt{\mathrm{Var}(g(\texttt{feat}))}\ , \\ &g(\texttt{feat}) + 1.96\ \sqrt{\mathrm{Var}(g(\texttt{feat}))}]\end{aligned} \tag{6.7}$$

where $\mathrm{Var}(g(\texttt{feat}))$ is calculated based on the distribution given in Equation 6.5.

We may consider the parameters to be different, within a 5% error rate, when the confidence intervals are not overlapped at all. Instead, we consider them to be stable whenever the confidence intervals overlap to some extent.

To this purpose, we devise the binary function overlap, which takes as input a generic pair of confidence intervals $\texttt{CIpair} = (\texttt{CI}_\alpha, \texttt{CI}_\beta)$ and returns either value 1 or 0, based on the overlap presence.

$$\textsc{overlap}(\texttt{CIpair}) = \begin{cases} 0 & \text{if } \texttt{CI}_\alpha \cup \texttt{CI}_\beta = \emptyset \\ 1 & \text{otherwise} \end{cases} \tag{6.8}$$

The comparison among confidence intervals is carried out separately for each variable. Chosen a certain feature, we check through the $g_1 ... g_m$ explainable models if the feature is relevant (coefficient different from 0). Whenever this happens, we build the confidence interval for the coefficient, using the function ConfInt, and we consider the set of all confidence intervals, namely $\mathcal{M}$, for the chosen variable.

Algorithm 5 Coefficients Stability Index (CSI)

Require: $g_1 ... g_m$
for feat in $\mathcal{F}$ **do**
  $\mathcal{M} = \{\}$
  for i in $1...m$ **do**
    if $g_i(\texttt{feat}) \neq 0$ **then**
      $\texttt{CI} = \textsc{ConfInt}\ (g_i(\texttt{feat}))$
      $\mathcal{M} = \mathcal{M} \cup \texttt{CI}$
  $n = 0$
  for CIpair in $C_2^{|\mathcal{M}|}$ **do**
    if overlap (CIpair) **then**
      $n++$
  $\textsc{Par}_{\texttt{feat}} = \frac{n}{\left|C_2^{|\mathcal{M}|}\right|}$
CSI = mean(Par) **return** CSI

We create all the possible combinations of the $\mathcal{M}$ items, two by two. This results in the set $C_2^{|\mathcal{M}|}$, whose generic element is $\texttt{CIpair} = (\texttt{CI}_\alpha, \texttt{CI}_\beta)$. We calculate the overlap between the two intervals, using the OVERLAP function, for all the pairs in $C_2^{|\mathcal{M}|}$.

The outcome is a count variable, which we normalise by dividing by the cardinality of the set $C_2^{|\mathcal{M}|}$. The value obtained ranges from 0 to 1 and it is called the Partial Index (PAR) for the variable considered. It represents a measure of concordance of the specific variable's coefficients among different LIME calls.
To achieve a general concordance metric, we average the Partial Indices of all the features and obtain the Coefficients Stability Index (CSI), ranging from 0 to 1. Consider now the index as a percentage, rescaling it from 0 to 100: the more CSI approaches 100, the more LIME coefficients may be considered stable in the neighbourhood of the chosen individual.

CSI steps are detailed in Algorithm 5.

6.3.0. Interpretation of the indices

The previously defined indices constitute a useful tool for assessing LIME stability in practical scenarios. By construction, VSI measures the concordance of the variables retrieved, whereas CSI tests the similarity among coefficients for the same variable, in repeated LIME calls.

Both of them range from 0 to 100.
High VSI values guarantee the variables retrieved in different LIME are almost always the same. On the contrary, low values testify explanations are not trustworthy: we may retrieve completely different variables explaining the same Machine Learning decision, according to different LIME calls.
As far as CSI is concerned, high values ensure LIME coefficient for each feature is reliable. Low values, instead, induce the practitioner to be very cautious: given a feature, the first LIME call will give back a certain value of the coefficient, but the one after is likely to retrieve a different value. Since the coefficient represents the impact of the feature on the Machine Learning decision, obtaining different values correspond to very different explanations.

Each index has a proper meaning and checks for a particular stability instance. Achieving high values for both of them ensures stability, however low values for only one metric are still possible. Keeping the measurements separated allows for understanding which one of the two complementary definitions of stability has been violated by the trained LIME method.

6.4.0. Practical Application on Credit Risk Data

We demonstrate the valuable information that the statistical indices provide to the user, by showing their results on the Credit Scoring dataset of Chapter 3.1.

We test LIME on several data points and we report LIME explanations for a "good" user, which has been correctly predicted by the GBM model, in Table 6.1. Different LIME settings are employed, to demonstrate how a wrong choice of parameters may yield inconsistent explanations and how the indices are able to correctly spot the instability.
To calculate the indices, LIME is applied 10 times, however the available implementation[2] allows to set the desired number of repetitions. Only the 7 most important features are considered in the explanation.

On the left, consistent explanations are achieved, testified by high stability values, whereas on the right a bad choice of the kernel width and the Ridge penalty values bring instability.

It is worth noticing LIME results for the stable explanation make sense from an economic and financial standpoint: the key regressors are the Credit Bureau Score (CBS), namely a comprehensive value developed using information provided by the Italian Credit Bureau, and the number of months when unpaid instalments occurred, within the last year. The user exhibits 0 months with unpaid instalments and falls inside a good class of CBS index. Such circumstances are the major ones leading Gradient Boosting model to classify him as a good payer.
On the contrary, the unstable LIME method produces different regression lines for each call, making it very hard to trust them since for the same individual we end up with totally different explanations.

On a 4 Intel-i7 CPUs 2.90GHz laptop, the indices took 10.23 and 11.54 seconds to be calculated for the left and right settings of Table 6.1 respectively.

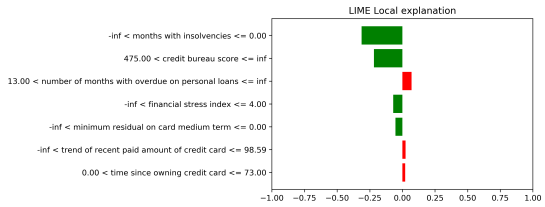
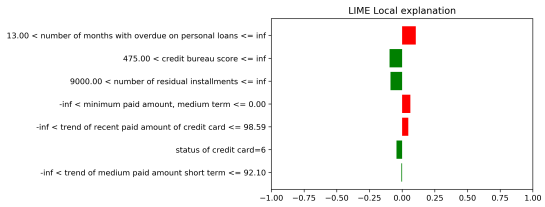
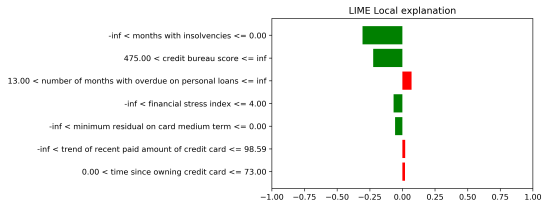
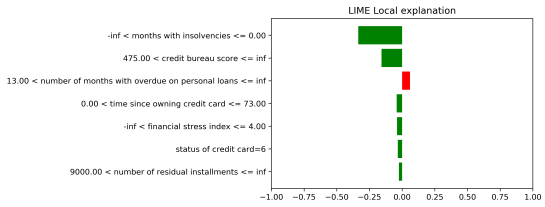
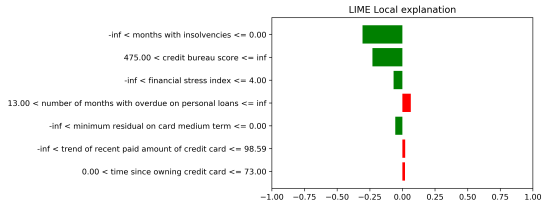
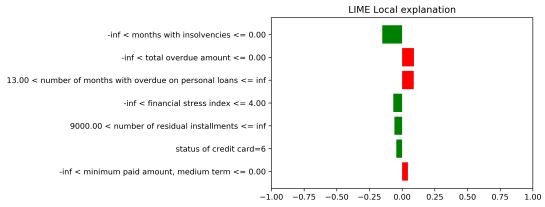
To sum up, we exploit the distribution of LIME local model coefficients, to build confidence intervals for each coefficient. The CSI stability index evaluates whether the intervals for the same variable among different LIME calls are similar. Meanwhile, we monitor whether the variables returned by different LIME calls are the same. This is done using another index: VSI. The two complementary indices both range from 0 to 100, where higher values correspond to a higher degree of stability.
An application to Credit Scoring data testifies how the indices can be used out-of-the-box on Tabular Data, although the stability framework can be applied also to images and text data, as long as LIME local model is chosen to be in the class of Ridge Regression models. In fact, the indices formulation relies on the Ridge model properties.

When used together, they provide useful insights to the practitioner about the consistency of the trained LIME method: they help understand whether LIME is likely to modify its output at the next call. We recommend always pairing LIME explanations with their stability indices values to prove that the explanation is stable, whereas the instability issue is rarely taken into account in business projects involving explainability. Even worse, often practitioners are either not aware of such drawback and they rely on a single LIME call, considering the explanation as reliable without further checks. In such cases, the

[2] https://github.com/giorgiovisani/LIME_stability

Table 6.1: LIME applied to Gradient Boosting model.

The sum of the bars' values, along with the intercept, produces the Local Ridge model prediction. The bars' length highlight the specific contribution of each variable: the green ones push the model towards "good payer" prediction, whereas the red ones to "bad payer".

Unit Number: 35 Kernel Width: 3 Ridge Penalty: 1 VSI: 89.44% CSI: 92.7%	Unit Number: 35 Kernel Width: 1.3 Ridge Penalty: 0.001 VSI: 14.17% CSI: 57.46%
	
	
	

instability issue is not spotted at all.

We consider the indices an important step, since it improves the trust in LIME as a stable explanation method. However, such result just ensures LIME is concordant among different applications: the model may still return explanations not really close to the Machine Learning model.

6.5.0. Extensive experiments on Stability Indices

To demonstrate that CSI and VSI Stability Indices behave as expected, we employ them on various combinations of open-source datasets, ML models and different LIME settings. In particular, we consider the Wine, Houses and Parkinson datasets and train a Gradient Boosting model (Xgboost), MultiLayer Perceptron with a single layer and Linear Regression. The stability indices have been recorded over 10 repetitions, requiring only the 5 most important variables in LIME explanations. The result is a condensed plot containing essnetial information of the 10 runs: each barplot represents the average coefficient of the specific variable over the 10 runs, while the black line records its standard deviation. Some of the plots contain more than 5 variables, meaning that some of the repeated explanations did not agree on the most important features -*this is reflected as well in the VSI value*-.

In the following Figures 6.1,6.2,6.3, the prediction of a specific unit from the Test set has been explained, using LIME with two different kernel width values per each ML model -*remaining LIME parameters are kept as default*-. We can appreciate how increasing the kernel width brings improved stability in general, both in a CSI and VSI perspective. A full discussion on the kernel width topic and how to set such hyper-parameter can be found in Chapter 7.

The main goal of the simulation is to visually show how bigger differences in important variables and related coefficients in repeated LIME explanations negatively affect the Stability Indices. Interesting to notice, also, that LIME hyper-parameters play an important role in the stability of the explanations, the kernel width above all.

In the following Figures 6.4, 6.5, instead, we exploit the same condensed plot to show how different units of the same dataset display substantially different LIME explanations and related stability. Again, a thorough discussion on the topic can be found in Chapter 7, while we currently focus on assessing the Indices goodness in spotting instability: we expect to witness lower CSI indices in correspondence of bigger standard deviations and lower VSI indices when more than 5 different variables are deemed important in the LIME repetitions.

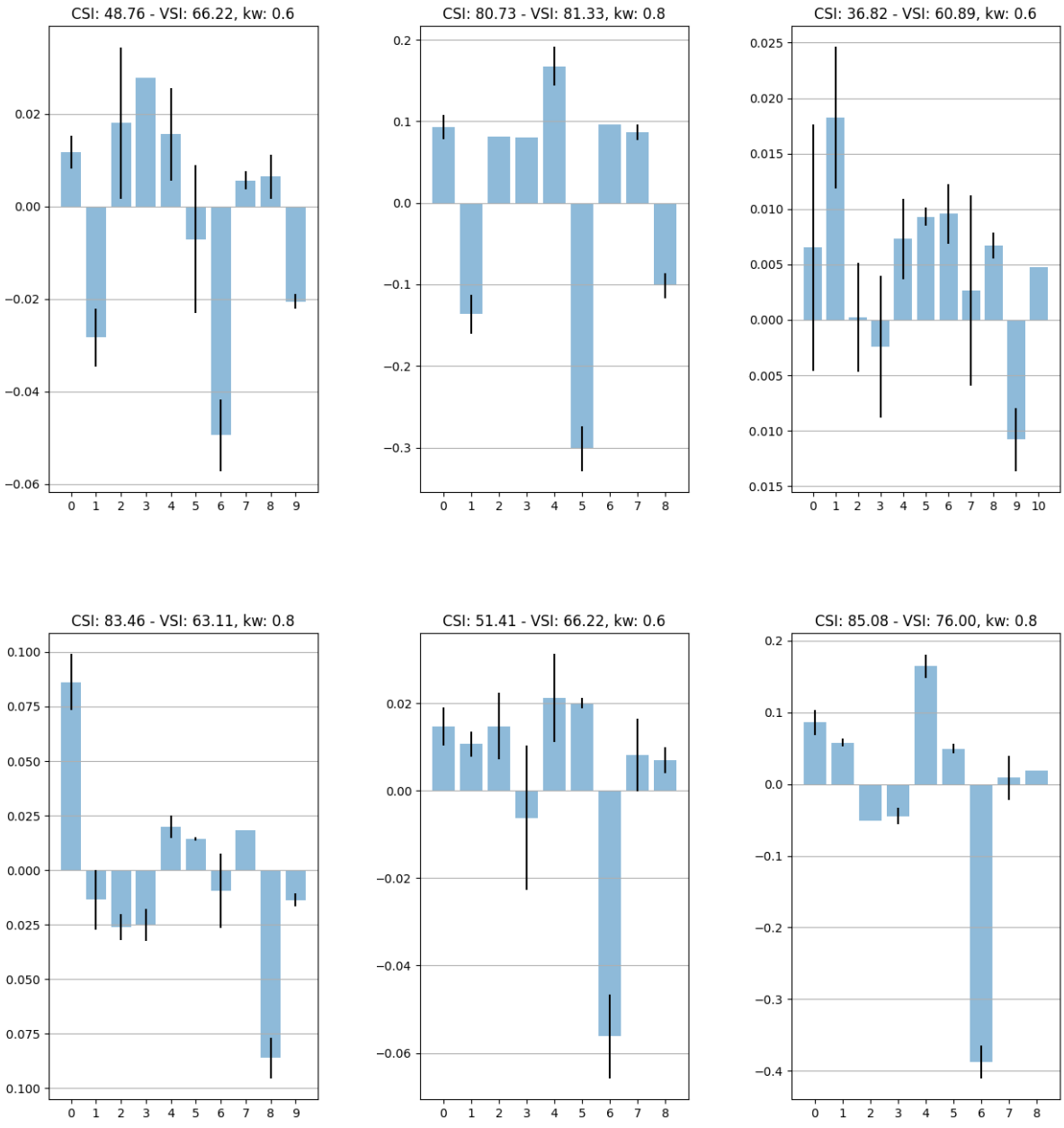


Figure 6.1: Analysis of LIME Stability on unit 3 of the Test data for ML models trained on the Wine dataset. Figures i) and ii) contain LIME coefficients for XGBoost model, Figures iii) and iv) are LIME explanations of a Neural Network, Figures v) and vi) display LIME coefficients on Linear Regression. Per each pair, LIME in the left-side picture has kernel width= 0.6, the right-side one has kernel width= 0.8

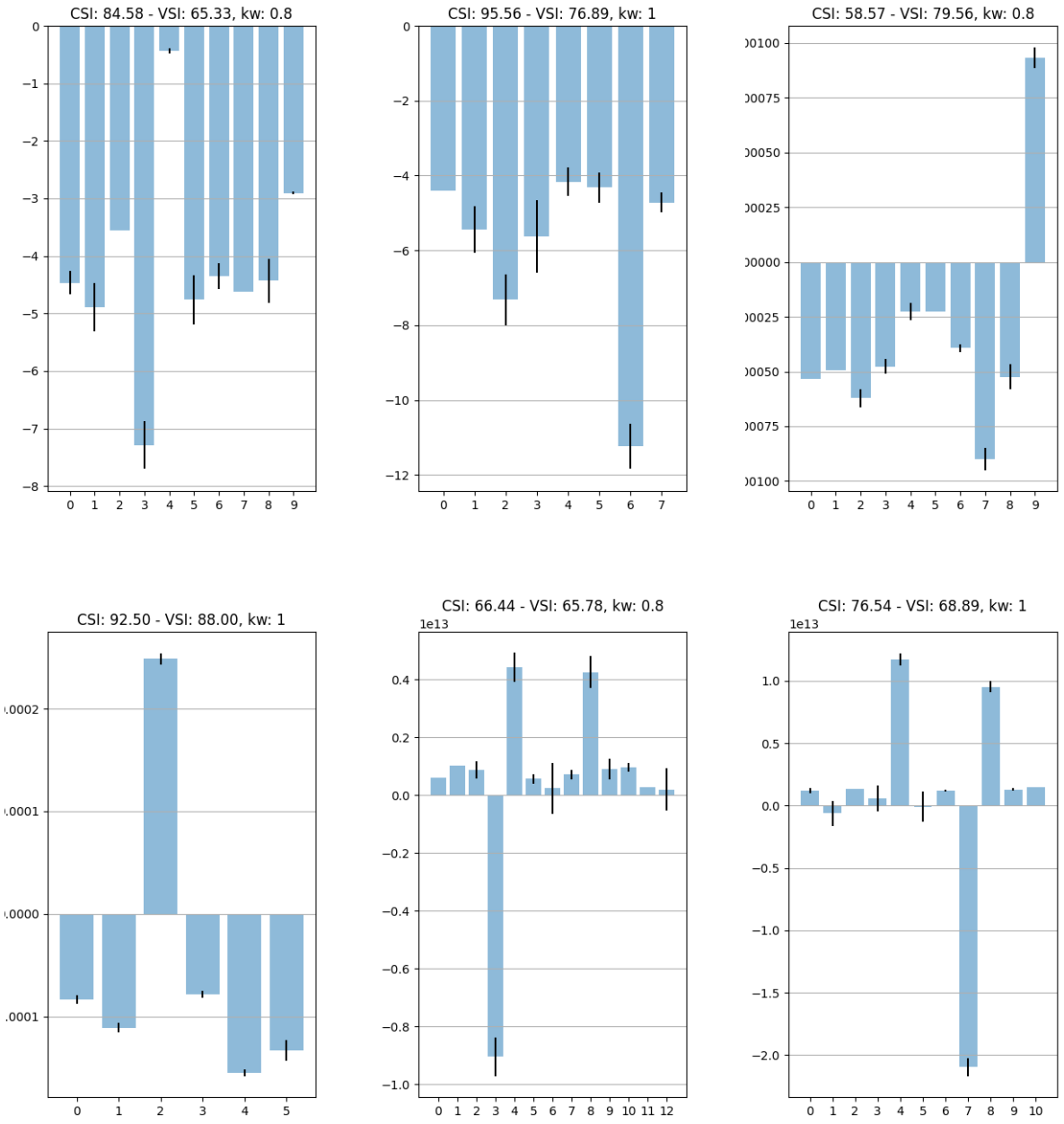


Figure 6.2: Analysis of LIME Stability on unit 12 of the Test data for ML models trained on the Houses dataset. Figures i) and ii) contain LIME coefficients for XGBoost model, Figures iii) and iv) are LIME explanations of a Neural Network, Figures v) and vi) display LIME coefficients on Linear Regression. Per each pair, LIME in the left-side picture has kernel width= 0.6, the right-side one has kernel width= 0.8

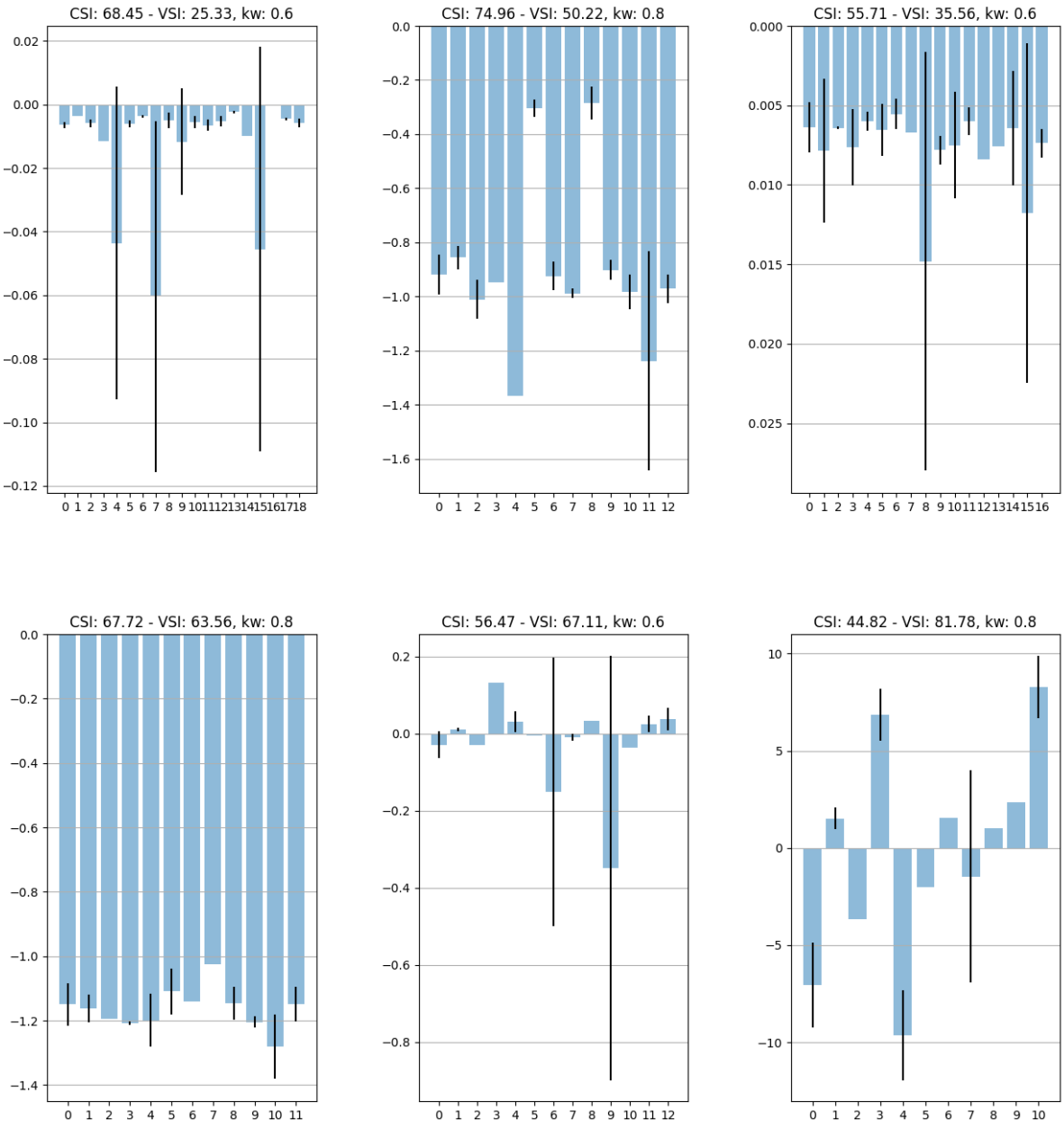


Figure 6.3: Analysis of LIME Stability on unit 6 of the Test data for ML models trained on the Parkinson dataset. Figures i) and ii) contain LIME coefficients for XGBoost model, Figures iii) and iv) are LIME explanations of a Neural Network, Figures v) and vi) display LIME coefficients on Linear Regression. Per each pair, LIME in the left-side picture has kernel width= 0.6, the right-side one has kernel width= 0.8

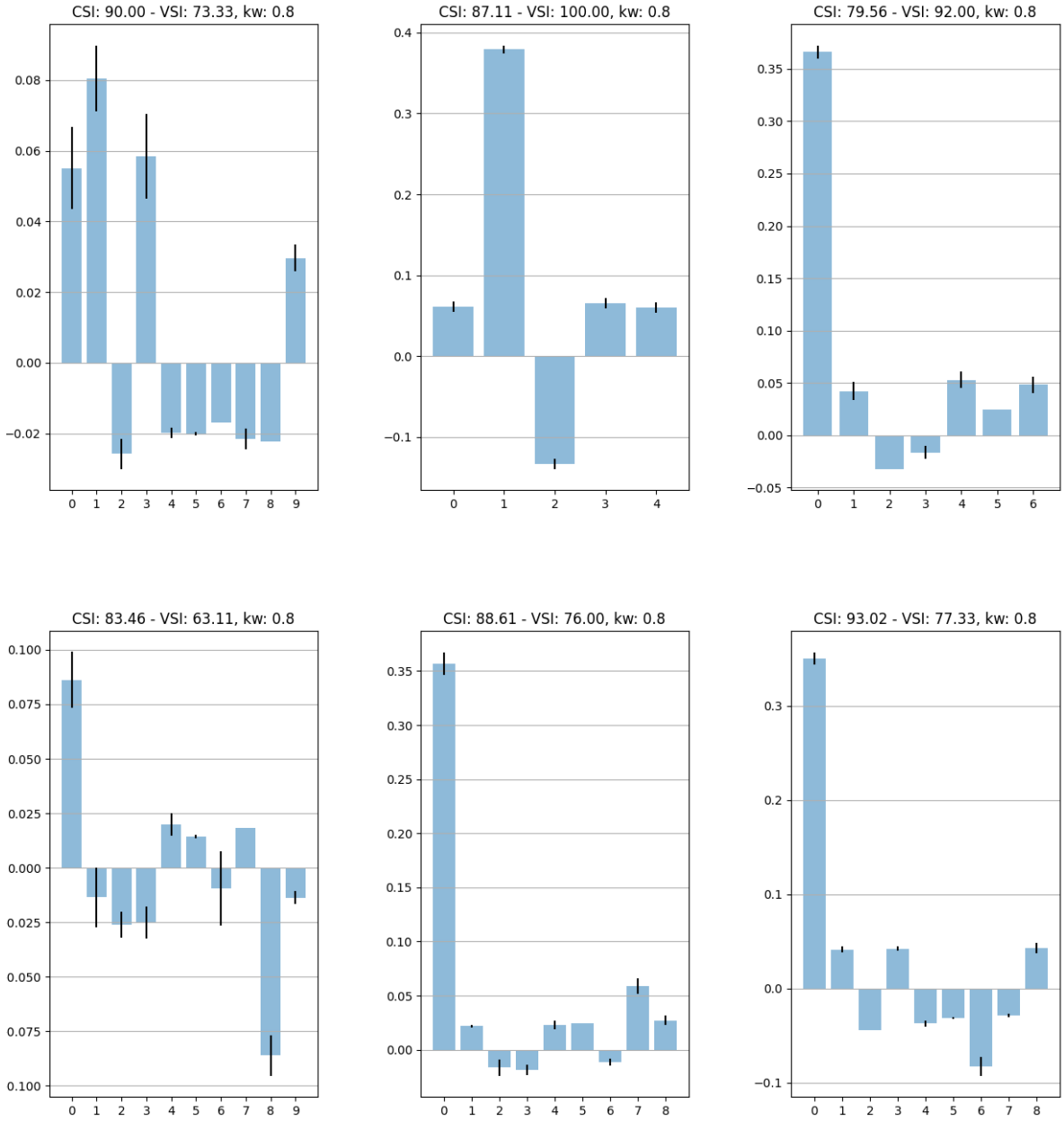


Figure 6.4: Analysis of LIME Stability on unit 1 to 6 of the Test data for Neural Network model trained on the Wine dataset. LIME explanations are carried out with kernel width= 0.8

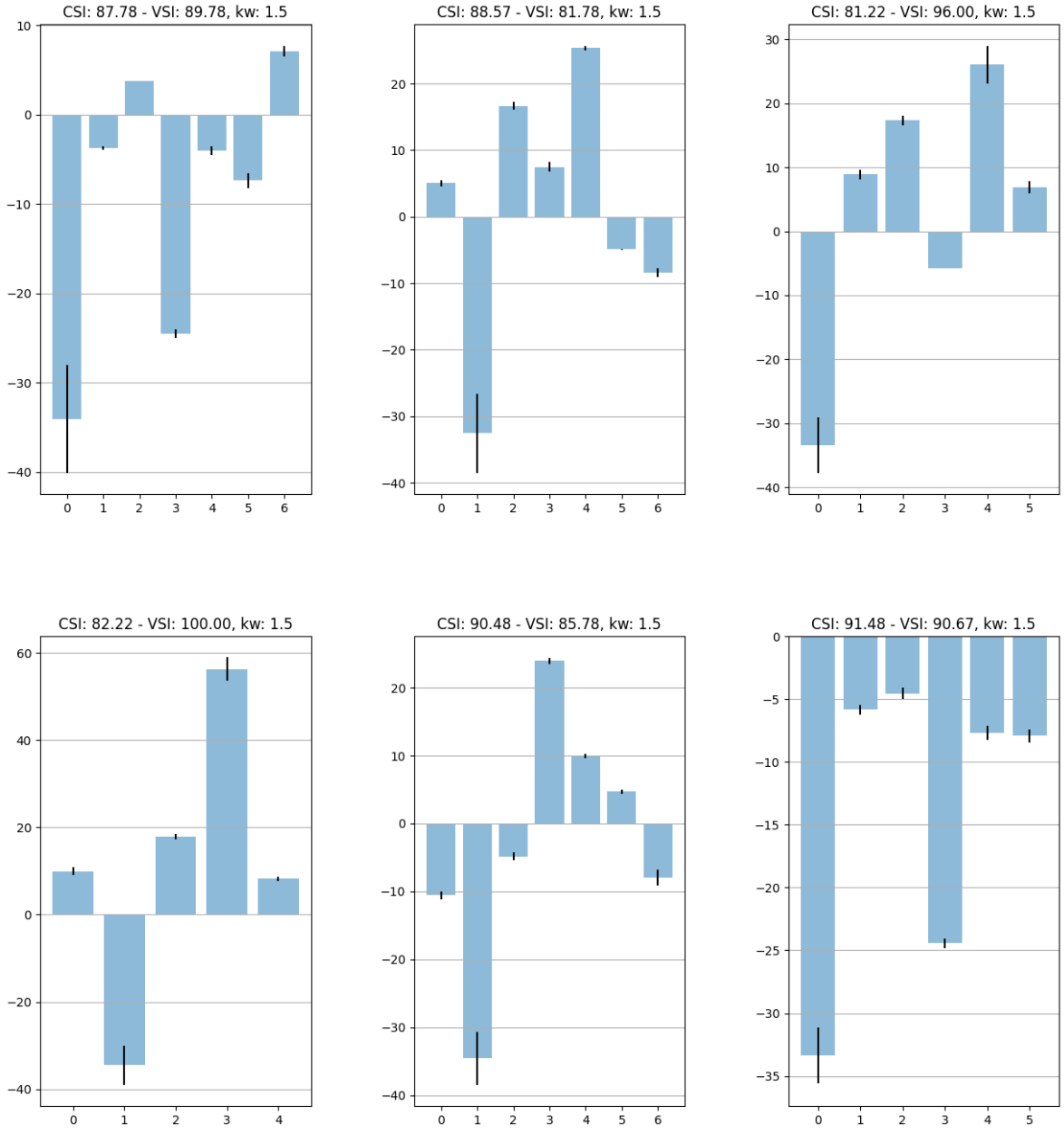


Figure 6.5: Analysis of LIME Stability on unit 1 to 6 of the Test data for XGBoost model trained on the Houses dataset. LIME explanations are carried out with kernel width= 1.5

Chapter 7

LIME: select the local nieghborhood size

As already explained in Chapter 5.2, the most important LIME hyper-parameter is the kernel width. This parameter governs the size of the neighbourhood influencing the π_{x_i} weights.

Recall that LIME weights are governed by Equation 5.1, which produces a kind of bell-shaped (gaussian) function, centred at $x^{(ref)}$. The weights π_{x_i} are assigned based on the $\ell 2$ distance of $\hat{x}^{(i)}$ from $x^{(ref)}$. If $\hat{x}^{(i)} \equiv x^{(ref)}$, then $\pi_{x_i} = 1$, while increasing the distance between the two points brings π_{x_i} closer to 0. Points at the same distance share the same weight.
The kernel width parameter, kw, controls the steepness of the bell-shaped function and behaves exactly like the variance parameter for a Gaussian pdf: small kw means a steeper curve and just a few points with significant weights π_{x_i}, increasing kw decreases the steepness of the weight function and extends the portion of $\hat{x}^{(i)}$ points with relevant weights.

Despite its importance, vanilla LIME does not take care of kw fine-tuning, on the contrary the method suggests a heuristic to set kw to a predefined value:

$$kw = 0.75\sqrt{d}$$

where d is the number of variables in the dataset.
Analyzing the heuristic, we notice that the kw value is the same for any $x^{(i)}$ to be explained, in fact it depends only on the dataset dimensionality. Although, as we stressed in Chapter 5.2, kw should be fine-tuned on every single datapoint to be explained, since it depends on the local non-linearity of f in the neighbourhood of $x^{(ref)}$.

At the same time, we have previously noticed how certain LIME hyper-parameter sets may yield unstable explanations. We want to achieve an explanation that is stable, but at the same time adheres to the ML model, i.e. locally resembles the f function.
To evaluate the stability we rely on the CSI and VSI indices, while the adherence is assessed using the R^2 statistic, which measures the goodness of the linear approximation through a set of points [37]. A high R^2 value indicates that the surrogate model is a good

approximation of the complex model in the local region, while a low R^2 value indicates that the surrogate model may not accurately capture the behaviour of the complex model in that region. All the figures of merit above span in the range $[0, 1]$, where higher values define respectively higher stability and adherence.

In the following, we explain how to balance the two most important properties of LIME explanations, by means of an accurate choice of the kw parameter.

7.1.0. Stability & Adherence Trade-off

From the theory, we have few helpful results to understand how the kernel width influences Stability and Adherence:

- Taylor Theorem [37] gives a polynomial approximation for any differentiable function, calculated in a given point. If we truncate the formula to the first degree polynomial, we obtain a linear function, its approximation error depends on the distance between the point in which the error is evaluated and the given point.
  Thus, if we assume the ML function to be differentiable in the neighbourhood of $x^{(ref)}$, the adherence of the linear model is expected to be inversely proportional to the width of the neighbourhood, i.e. to the kernel width. This is true since the approximation error depends on the distance from the two points, namely the neighbourhood size.

- in Linear Regression, the standard deviation of the coefficients is inversely correlated to the standard deviation of the X variables [37].
  The stability of the explanations depends on the spread of the X variables in our weighted dataset. We then expect the kernel width and Stability to be directly proportional.

To illustrate the conjectures above, we run LIME for different kernel width values and evaluate both R^2 and CSI metrics (VSI is not considered in the Toy Dataset, since only one variable is present). In Figure 7.1 the results of such an experiment, for the reference unit, are shown.

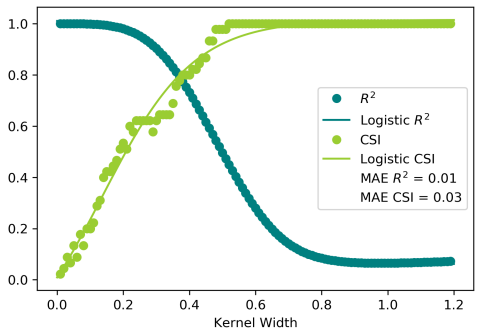


Figure 7.1: Relationship among kernel width, R^2 and CSI

Both the adherence and stability are noisy functions of the kernel width: they contain some stochasticity, due to the different datasets generated by each LIME call. Despite

this, it is possible to detect a clear pattern: monotonically increasing for the CSI Index and monotonically decreasing for the R^2 statistic.
For numerical evidence of these properties, we fit the Logistic function [116], which retrieves the best monotonous approximation to a set of points. The goodness of the logistic approximation is confirmed by a low value of the Mean Absolute Error (MAE).
To corroborate our assumption, the same process has been repeated on all the units of the Toy Dataset, obtaining average MAE for the R^2 approximation of 0.005 and for the CSI of 0.026. The logistic growth rate has also been inspected: R^2 highest growth rate is -10.78 and CSI lowest growth rate is 7.20. These results ensure the monotonous relationships of adherence and stability with the kernel width, respectively decreasing and increasing.

7.2.0. OptiLIME

Previously, we empirically showed that adherence and stability are monotonous noisy functions of the kernel width: for increasing kernel width we observe, on average, decreasing adherence and increasing stability.
Our proposition consists in a framework which enables the best choice for the trade-off between stability and adherence to the explanations. OptiLIME sets a desired level of adherence and finds the largest kernel width, matching the request. At the same time, the best kernel width provides the highest stability value, constrained to the chosen level of adherence. At the end of the day, OptiLIME consists of an automated way of finding the best kernel width. Moreover, it empowers the practitioner to be in control of the trade-off between the two most important properties of LIME Local Explanations.

To retrieve the best width, OptiLIME converts the decreasing R^2 function into $l(kw, \tilde{R}^2)$, by means of Formula 7.1:

$$l(kw, \tilde{R}^2) = \begin{cases} R^2(kw), & \text{if } R^2(kw) \leq \tilde{R}^2 \\ 2\tilde{R}^2 - R^2(kw) & \text{if } R^2(kw) > \tilde{R}^2 \end{cases} \tag{7.1}$$

where $\tilde{R}^2$ is the requested adherence.
The function $l(kw, \tilde{R}^2)$ presents a global maximum in correspondence of $\tilde{R}^2$, chosen by the practitioner. We are particularly interested in the $\arg\max_{kw} l(kw, \tilde{R}^2)$, namely the best kernel width.
In order to solve the optimum problem, Bayesian Optimization is employed, since it is the most suitable technique to find the global optimum of noisy functions [76]. The technique relies on two parameters to be set beforehand: p, number of preliminary calls with random kw values, m, number of iterations of the search refinement strategy. Increasing the parameters ensures finding a better kernel width value, at the cost of longer computation time.

In Figure 7.2, an application of OptiLIME to the reference unit of the Toy Dataset is presented. $\tilde{R}^2$ has been set to 0.9, $p = 20$ and $m = 40$. The points in the plot represent the distinct evaluations performed by the Bayesian Search in order to find the optimum. Comparing the plot with Figure 7.1, we observe the effect of Formula 7.1 on the left part of the R^2 and $l(kw, \tilde{R}^2)$ functions. In Figure 7.2 the search has converged to the maximum,

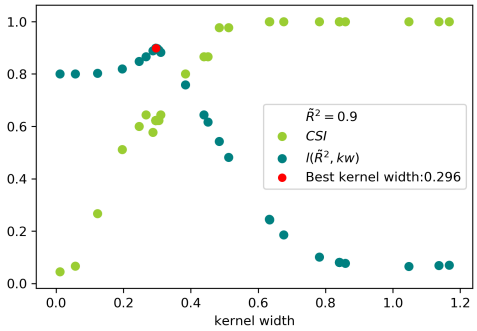


Figure 7.2: OptiLIME Search for the best kernel width

evaluating various points close to the best kernel width. At the same time, it is evident the stochastic nature of the CSI function: the several CSI measurements, performed in the proximity of 0.3 value of the kernel width, show a certain variation. Nonetheless, it is possible to recall the increasing CSI trend.

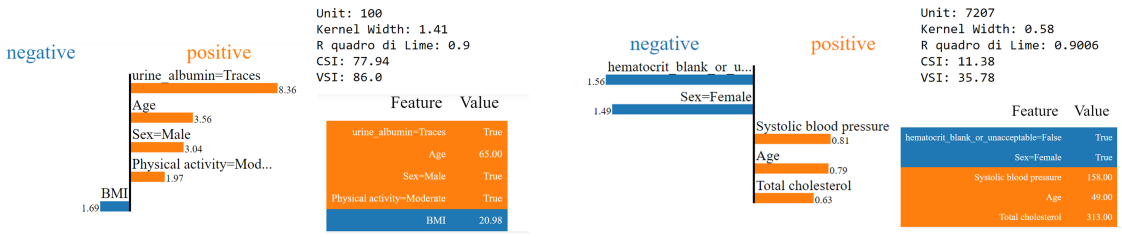
7.3.0. OptiLIME Application to Real-World Datasets

NHANES Medical Data

We now test the OptiLIME framework on the NHANES medical dataset, described in Chapter 3.2. Recall that model prediction consists of the hazard ratio for each individual: higher prediction means the individual is likely to be alive for a shorter period of time. Therefore, positive variables coefficients define risk factors, whereas protective factors have negative values.

After the ML training phase, we want to achieve the optimal explanations of the XGBoost model on specific units of the dataset. We do so using the OptiLIME framework on two randomly chosen individuals and we visually show the results. In our simulation, we consider 0.9 as a reasonable level of adherence. OptiLIME is employed to find the proper kernel width to achieve R^2 value close to 0.9 while maximizing stability indices for the local explanation models.
We recall once again that Local linear surrogates (such as LIME and OptiLIME) have similar model interpretation as Linear Regression models, but that is valid only for small changes in the original variable values. This also means that the same variable may have a different impact on two individuals if they live in two distant parts of the variables space $\mathcal{X}$. As an example, the Age variable has a different impact on the individuals in Figure 7.3: having 1 year more for Unit 100 (increasing from 65 to 66 years) will raise the death risk of 3.56 base points, for Unit 7207 1 year of ageing (from 49 to 50) will increase the risk of just 0.79. Another example is the impact of Sex, which is more pronounced in elder people: being female is a protective factor for 1.49 points at age 49, at age 65 being male has a much worse impact, as a risk factor for 3.04.

For Unit 100 in Figure 7.3a, the optimal kernel width is a bit higher compared with Unit



(a) Best LIME Explanation, Unit 100 (b) Best LIME Explanation, Unit 7207

Figure 7.3: NHANES individual Explanations using OptiLIME

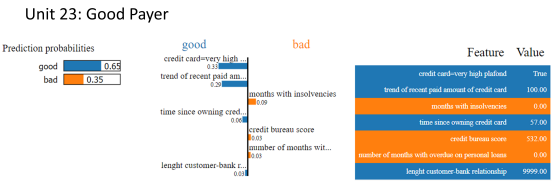
7207 in Figure 7.3b. This is probably caused by the ML model having a higher degree of non-linearity for the latter unit: to achieve the same adherence, we are forced to consider a smaller portion of the ML model, hence a small neighbourhood. Smaller kernel width implies also a reduced Stability, testified by small values of the VSI and CSI indices. Whenever the practitioner desires more stable results, it is possible to re-run OptiLIME with a less strict requirement for adherence. It is important to remark that low degrees of adherence will make the explanations increasingly more global: the linear surface retrieved by LIME will consist of an average of many local non-linearities of the ML model.

The computation time largely depends on the Bayesian Search, controlled by the parameters p and m. In our setting, $p = 10$ and $m = 30$ produce good results for both the units in Figure 7.3. On a 4 Intel-i7 CPUs 2.50GHz laptop, the OptiLIME evaluation for Unit 100 and Unit 7207 took respectively 123 and 147 seconds to compute. For faster, but less accurate results, the Bayesian Search parameters can be reduced.

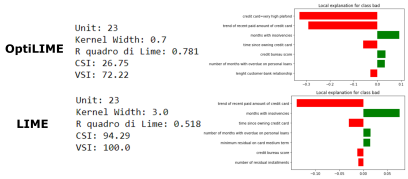
Credit-Scoring Data

We also apply the OptiLIME framework to obtain explanations of the Xgboost model f trained on the Credit-Scoring dataset as described in 3.1. In particular, we compare OptiLIME and LIME explanations in terms of stability and reliability and business meaning, on two reference units: i) Unit 23 classified as "Good Payer" by f, and ii) Unit 1746 classified as "Bad Payer".

Comparing Figures 7.4,7.5 we notice how LIME assigns the same kernel width to both units, even if they are quite different individuals (they sit on very different regions of the $\mathcal{X}$ manifold). Since LIME explanation considers a larger neighbourhood than OptiLIME, it has a more global flavour approximating f on a larger region and averaging out various non-linear regions. This results in a poor adherence of the LIME explanation to f on the locality of the points, testified by a low value of R^2, but at the same time it achieves higher stability. OptiLIME instead, provides us with a predefined level of adherence (set as $R^2 = 0.8$).

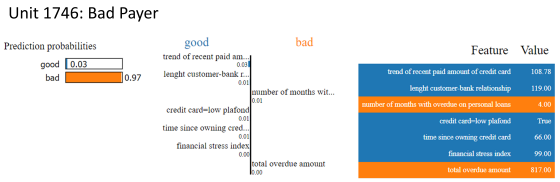


(a) f prediction and salient features values

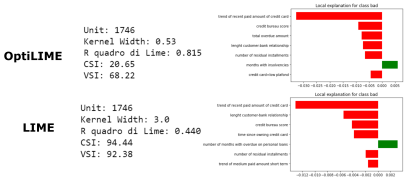


(b) LIME and OptiLIME explanations

Figure 7.4: Individual Explanations using OptiLIME for unit 23 (Good Payer)



(a) f prediction and salient features values



(b) LIME and OptiLIME explanations

Figure 7.5: Individual Explanations using OptiLIME for unit 1746 (Bad Payer)

To sum up, it is difficult to tune properly LIME's main parameter: different values of the kernel width provide substantially different explanations.
We notice how smaller kernel width values provide a more adherent LIME plane to the ML surface, i.e. a more realistic local explanation. At the same time, training Linear Regression models on small neighbourhoods cause higher model variance and instability, by design. The trade-off between the adherence and stability properties is extremely valuable since it empowers the practitioner to choose the best kernel width consciously.

We exploit these findings in order to tackle LIME weak points. The result is the OptiLIME framework, which achieves stability of the explanations and automatically finds the proper kernel width value, according to the practitioner's needs. The framework may serve as an extremely useful tool: through OptiLIME, the practitioner knows how much he can trust the explanations, based on their stability and adherence values. Nonetheless, we acknowledge that the optimization framework may be improved to allow for a faster and

more precise optimum solution.

Chapter 8

Bridging the Gap between Local and Global Explanations

Throughout this thesis, we made a clear distinction between local and global explanation techniques and we pointed out the pros and cons of both methodologies. In general global techniques are more desirable, since they would empower the practitioner with the understanding of the whole complex f model. Local explanations would be simply specific use-cases of a reliable and clear global explanation. Unfortunately, existing global techniques do not usually meet the quality standards being either too complex to understand or too simple and not enough reliable.

Given the difficulty in overcoming the intrinsic trade-off between clarity and fidelity of global explanations, the idea of combining the two frameworks in a single unified one has recently become an interesting and new topic in the XAI field.
There are several benefits to combining local and global explanation methods:

1. Comprehensive understanding: Local explanation methods provide detailed information about how a specific decision was made, while global explanation methods provide an overview of the model's behaviour as a whole. Combining the two can give a more comprehensive understanding of the model's decision-making process.

2. Improved trust and transparency: By providing both local and global explanations, it is possible to increase the transparency of f and build trust with stakeholders.

3. Increased robustness: Combining multiple (local) explanations can help identify potential weaknesses or biases in model, allowing the practitioner to improve its robustness.

4. Better decision-making: By understanding both the local and global behaviour of f, it is possible to make more informed decisions about how to use it.

One of the first attempts in this direction dates back to Lundberg and Lee [78] through the local SHAP values and their ***additivity*** property, which guarantees to obtain reliable global feature attributions by averaging them over the entire dataset. Unfortunately vanilla SHAP is too computationally intensive and it's not affordable to estimate it on each

unit of very large -*and possibly high-dimensional*- datasets, so the real chance to compute global SHAP values came with TreeSHAP [77], which provides a fast algorithm valid for Tree-based models only.
Ribeiro, Singh, and Guestrin [92] propose Anchors, a method extracting local subsets of the features (anchors) that are sufficient to recover the same prediction, while having good global coverage. This can be considered as a way to create accurate explanations which are in between the local and global worlds, i.e. they push to be as widely applicable as possible, remaining accurate at the same time.
Let us also mention Setzu et al. [101], which proposes to ***aggregate local explanations***: starting from local decision rules, GlocalX combines them in a hierarchical manner to form a global explanation of the model. In the *ad hoc* setting, Harder, Bauer, and Park [48] proposes to train an interpretable and differentially private model by using several local linear maps per class: the pseudo-probability of belonging to a given class is the softmax of a mixture of linear models. This approach is limited to the classification setting, as with GlocalX and Anchors.

8.1.0. GLEAMS

We propose ***GLEAMS*** (Global & Local ExplainAbility of black-box Models through Space partitioning), a ***post-hoc*** interpretability method providing both global and local explanations. Our goal is to build a ***global surrogate*** $\hat{f}$ of f that shares the global shape of the black-box model f but, at the same time, has a simple local structure. Our main motivation in doing so is to extract simultaneously from $\hat{f}$: i) local explanations for any instance, ii) overall importance of the features, iii)"what-if"-type explanations of any given scenario (not restricted to a local modification), and iv) visual summaries.

In more detail, the core idea of GLEAMS is to recursively split the input space into rectangular cells on which the black-box model can be well-approximated by linear models. This is done in a two-step process. The first step is to sample N points $\hat{x}^{(1)}, \dots, \hat{x}^{(N)}$, evenly spread in $\mathcal{X}$. For each of these points, we query f to obtain $\hat{y}^{(i)} = f(\hat{x}^{(i)})$. Thus we effectively create a dataset $(\hat{x}^{(i)}, \hat{y}^{(i)})$, which we call ***measurement points***. A somewhat hidden assumption here is that we can query f as much as desired, in constant time.
Subsequently, the idea is to recursively split the input space $\mathcal{X}$ and to approximate the model f by a ***linear model*** on each cell of the partition until the linear model in each rectangular leaf fits the measurement points with sufficiently good accuracy. This process is summarized in Figure 8.1.

GLEAMS stands back from other hybrid approaches combining Tree-based models and Linear Regression, such as Zhang, Nettleton, and Zhu [124] and Ilic et al. [60], since it proposes to clearly split the domain in mutually exclusive partitions and train separate linear models on each of them. This produces a highly interpretable final model, providing only one model to query based on the region where a specific point falls. On the contrary, the aforementioned models exploit a combination of linear models and tree-based models to obtain a good fit over the entire domain, without comparting the space in smaller regions equipped with simple models. Instead, they produce a single complex model valid

on the entire domain space, which somewhat hinder interpretability.

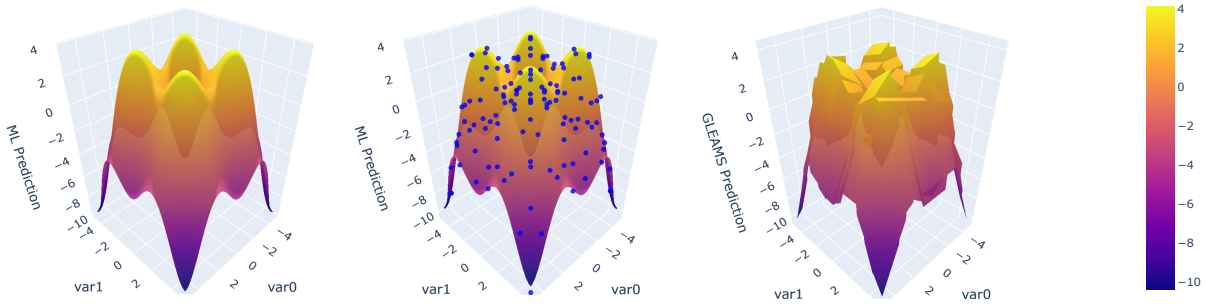


Figure 8.1: Overview of the global surrogate model construction. ***Left panel:*** the black-box model maps the input space (here $\mathcal{X} = [0,1]^2$) to $\mathbb{R}$, which we can visualize as a surface. ***Middle panel:*** we generate N Sobol points on $\mathcal{X}$, giving rise to N measurement points on the surface (in blue). ***Right panel:*** we fit a piecewise-linear global surrogate model $\hat{f}$ on the measurement points by recursively splitting $\mathcal{X}$.

The local explanations are then simply given by the feature coefficients of these local surrogate models, while global indicators can be readily computed from those, without querying f model further. Intuitively, given a partition with not too many cells, it is convenient for the user to look at individual features and visualize globally the model as a piecewise-linear function. Concurrently, any new example to explain will fall into a cell, and the user can get a local explanation solely by looking at the coefficients of the local linear model.

We show experimentally that GLEAMS compares favourably with existing methods in Chapter 8.2.

8.1.1.0. Measurement points

Our main assumption, going into the description of the method, is the following:

Assumption 1. The input space $\mathcal{X}$ is a hyper-rectangle of $\mathbb{R}^d$. That is, there exist real numbers $a_j, b_j \in \mathbb{R}$ for any $j \in [d]$ such that

$$\mathcal{X} = \prod_{j=1}^{d} [a_j, b_j] \,.$$

Intuitively, this corresponds to a situation where each variable has a prescribed range (for instance, age takes values in $[0, 120]$). In particular, we assume that further examples to explain all fall within this range. If they do not, we attribute to these points the explanations corresponding to the projection of these points on $\mathcal{X}$. Note that we do not assume normalized data: a_j and b_j can be arbitrary. In practice, the boundaries are either inferred from a training set, or directly provided by the user.
A consequence of Assumption 1 is that we can easily find a set of points covering $\mathcal{X}$ efficiently. To achieve this, we use ***Sobol sequences*** [104]. In a nutshell, a Sobol sequence

is a quasi-random sequence of points $\hat{x}^{(1)}, \ldots, \hat{x}^{(N)}$ filling the unit cube $[0,1]^d$ as N grows. We demonstrate this in dimension $d = 2$ in Figure 8.2.

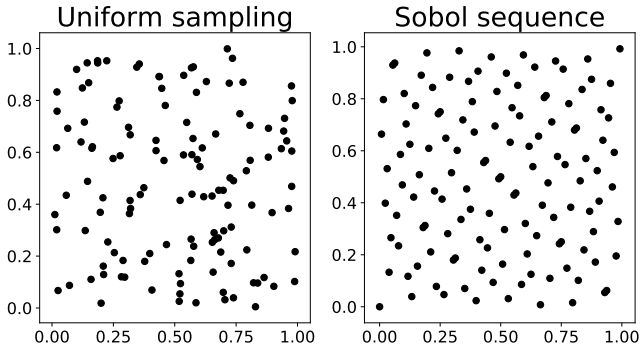


Figure 8.2: Differences between uniform sampling (***Left panel***) and Sobol sequence (***Right panel***) in dimension 2. The discrepancy between points is much lower, while maintaining some apparent randomness in the sampling.

There are several reasons why we use Sobol sequence as the sampling scheme of GLEAMS. First, we want to be able to provide explanations for the whole input space $\mathcal{X}$, thus the measurement points should cover $\mathcal{X}$ as N grows (there should be no part of $\mathcal{X}$ that remains unprobed). Second, the decision surface of the black-box model can be quite irregular: good predictive accuracy usually means that the model has a lot of local variation. Thus, to recover this local variability, the measurement points need to have discrepancy as low as possible. Sobol points have a lower discrepancy than points sampled uniformly at random. Third, there is still some irregularity in the sampling process, preventing artifacts appearing from a simple grid sampling. This is especially important since the points are later used as input for linear models: we want them to be un-correlated. Finally, even though other pseudo-random sequences exist in the literature such as Halton sequences [47], we found Sobol sequences to be the fastest to generate and the most reliable in high dimensional settings, i.e. do not present significant variable correlations.
To conclude with the description of the sampling process, we note that a consequence of the curse of dimensionality is the need to take a number of points that is exponential in the dimension of $\mathcal{X}$. Namely, in our experiments, we set $N = 2^{d+c}$, where d is the dimension of $\mathcal{X}$ and c is a numerical constant taken as high as possible with respect to our computing power.

8.1.2.0. Splitting criterion

A very convenient way to partition the input space is to recursively split $\mathcal{X}$, cutting a cell if some numerical criterion is satisfied. Additionally, we consider splits that are parallel to the axes. The main reason for doing so is the ***tree structure*** that emerges, allowing later on for fast query time of new inputs (linear in the depth of the final tree). In this Chapter, we describe the criterion used by GLEAMS.
A popular approach is to fit a constant model on each cell [10, CART]. We see two problems with constant fits. First, this tends to produce large and thus hard to interpret trees (see, ***e.g.***, Chan and Loh [14]). Second, the local interpretability would be lost,

since the local models would not depend on the features in that case. Thus we prefer to fit linear models on each leaf, and our numerical criterion should indicate, on any given leaf, if such simple model is a good fit for f.

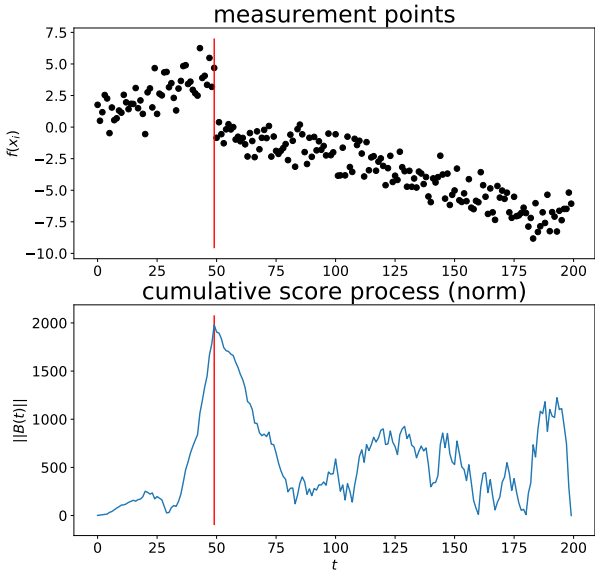


Figure 8.3: Evolution of the norm of the cumulative score process. ***Top panel:*** measurements of a piecewise-linear model (dark dots) visualized along one axis. ***Bottom panel:*** evolution of $\|B(t)\|_1$ as a function of t (solid blue line). The process presents a clear maximum, which gives us a candidate split for this axis (vertical red line).

GLEAMS relies on a variant of model-based recursive partitioning [122, MOB]. Intuitively, any well-behaved function can be locally approximated by a linear component, yielding a statistical model which we now describe. On any given leaf, for any given feature j, we can reorder the measurement points such that the points contained in the leaf are $\hat{x}^{(1)}, \ldots, \hat{x}^{(n)}$, with $\hat{x}_j^{(1)} \leq \ldots \leq \hat{x}_j^{(n)}$. To these points correspond model values $\hat{y}^{(1)}, \ldots, \hat{y}^{(n)}$. In accordance with the intuition given above, we write $\hat{y}^{(i)} = \beta^\top \tilde{x}^{(i)} + \mathcal{E}_i$ for all $i \in [n]$, where $\beta \in \mathbb{R}^{d+1}$ are the coefficients of our local linear model, $\tilde{x}^{(i)} \in \mathbb{R}^{d+1}$ is the concatenation of $\hat{x}^{(i)}$ with a leading 1 to take the intercept into account, and $\mathcal{E}_i$ is an error term. Let us now follow Merkle and Zeileis [80] and make an i.i.d. Gaussian assumption on the $\mathcal{E}_i$s, giving rise to the likelihood

$$\mathcal{L}(\beta; \hat{x}^{(i)}) = \frac{1}{(2\pi\sigma^2)^{d/2}} \exp\left(\frac{-1}{2\sigma^2}(\hat{y}^{(i)} - \beta^\top \tilde{x}^{(i)})^2\right), \tag{8.1}$$

where $\sigma^2 > 0$ is the variance of $\mathcal{E}_i$ for all $i \in [n]$. Taking the log in Eq. (8.1), the log-likelihood of a single observation is given by

$$\ell(\beta; \hat{x}^{(i)}) = \frac{-1}{2}\left\{\frac{1}{\sigma^2}(\hat{y}^{(i)} - \beta^\top \tilde{x}^{(i)})^2 + d \log 2\pi\sigma^2\right\}. \tag{8.2}$$

The individual scores are obtained by taking the partial derivatives of the log-likelihood

with respect to the parameters, that is, for any $i \in [n]$,

$$s(\beta; \hat{x}^{(i)}) = \left(\frac{\partial \ell(\beta; \hat{x}^{(i)})}{\partial \beta_0}, \frac{\partial \ell(\beta; \hat{x}^{(i)})}{\partial \beta_1}, \dots, \frac{\partial \ell(\beta; \hat{x}^{(i)})}{\partial \beta_d} \right)^\top . \tag{8.3}$$

A straightforward computation yields

$$s(\beta; \hat{x}^{(i)}) = \left(\hat{y}^{(i)} \cdot \tilde{x}^{(i)} - \tilde{x}^{(i)} \tilde{x}^{(i)^T} \beta \right) \sigma^{-2} . \tag{8.4}$$

The maximum likelihood estimate $\hat{\beta}$ is obtained by solving $\sum_i s(\beta; \hat{x}^{(i)}) = 0$: in this case this coincides with the ***ordinary least-squares*** estimate, in a sense the best linear fit on the entire leaf. However, what we want is to split this leaf if the deviations from the linear model computed on the whole leaf are too strong. Therefore, we compute the ***cumulative score process*** defined by

$$\forall t \in [0,1], \quad B_j(t; \hat{\beta}) = \frac{1}{\sqrt{n}} \sum_{i=1}^{\lfloor nt \rfloor} s(\hat{\beta}; \hat{x}^{(i)}) , \tag{8.5}$$

and take as a criterion *the maximum of the L^1 norm of $B_j(t)$, across all features.* The computation of the best split is described in Algorithm 6 and we provide an example in Figure 8.3.

Algorithm 6 `GetBestSplit`: Get the best split for a given leaf.

Require: $\hat{x}^{(1)}, \dots, \hat{x}^{(n)}$: Sobol points in current leaf; $\hat{y}^{(1)}, \dots, \hat{y}^{(n)}$: corresponding f values

for $j = 1$ to d **do**
  re-order $\hat{x}^{(1)}, \dots, \dots, \hat{x}^{(n)}$ according to feature j
  re-order $\hat{y}^{(1)}, \dots, \hat{y}^{(n)}$ accordingly
  compute $\hat{\beta} = \mathrm{OLS}(\hat{Y}; \hat{X})$
  compute predictions $y_{ML}^{(i)}$
  set $\sigma^2 = \frac{1}{n} \sum_{i=1}^n (\hat{y}^{(i)} - y_{ML}^{(i)})^2$
  compute scores $s(\hat{\beta}; \hat{x}^{(i)})$ according to Eq. (8.4)
  compute B_j according to Eq. (8.5)
  store $m_j = \max_{t \in [0,1]} \|B_j(t)\|_1$
  store $t_j = \arg\max_{t \in [0,1]} \|B_j(t)\|_1$
return $j^\star = \arg\max_{j \in [d]} m_j$, feature to split
return $t_j^\star$, value of the split
return $\hat{\beta}$, local linear model

Note that Eq. (8.5) corresponds to Eq. (13) in Merkle and Zeileis [80], with the notable difference that Eq. (13) is normalized (by the square root of the estimated covariance matrix of the scores). We observe empirically that normalizing yields worse results when detecting the splits.

8.1.3.0. Global surrogate model

To build the global surrogate model $\hat{f}$, starting from the hyper-rectangle $\mathcal{X}$, we *iteratively split along the best dimension* following the strategy described in the previous Chapter. This process is stopped once one of two criteria is met. On the one side, we stop splitting when leaves have a coefficient of determination which is high enough: in that event, the linear fit on the leaf is good enough and there is no need to split further. In practice, we observe that $R^2 > .95$ is a reasonable stopping criterion. On the other side, one needs a minimum number of points to be able to fit a linear model in dimension d, thus GLEAMS also stops splitting if the number of points in the leaf is less than a threshold $n_{\min}$. In practice, we set $n_{\min} = \min(20, d+1)$. This recursive procedure is described in Algorithm 7, and an example output is showcased in Figure 8.4.

Algorithm 7 `RecTree`: Recursive construction of $\hat{f}$.

Require: R: hyper-rectangle; S: Sobol points inside R; V: values of f on S
initialize $T \leftarrow \mathcal{X}$, tree storing partition and models
for $L \in T$.leaves **do**
  if $R^2(L) \leq \rho$ and $n(L) \geq 2n_{\min}$ **then**
    set $\hat{x} = \{\hat{x}^{(1)}, \dots, \hat{x}^{(n)}\} = S \cap L$
    set $\hat{y} = \{\hat{y}^{(1)}, \dots, \hat{y}^{(n)}\}$ accordingly
    $(j, t_j) =$ `GetBestSplit`$(\hat{x}, \hat{y})$
    $L^\ell \leftarrow L \cap \{\hat{x} : \hat{x}^{(i)}j \leq t_j\}$
    $L^r \leftarrow L \cap \{\hat{x} : \hat{x}^{(i)}j > t_j\}$
    L.left_child $\leftarrow$ `RecTree`(L^ℓ, S, V)
    L.right_child $\leftarrow$ `RecTree`(L^r, S, V)
return T

The construction of the global surrogate model, $\hat{f}$, is then achieved by simply sampling Sobol points on $\mathcal{X}$, computing the value of the black-box model at these points, and then using Algorithm 7. This is summarized in Algorithm 8.

Algorithm 8 `GlobalSurrogate`: Compute the global surrogate model used by GLEAMS.

Require: f: black-box model; $\mathcal{X}$ or its boundaries; N: number of Sobol points; ρ: R^2 threshold; $n_{\min}$: min. number of points per leaf
generate $S = \{\hat{x}^{(1)}, \dots, \hat{x}^{(N)}\}$ Sobol points in $\mathcal{X}$
compute $\hat{y}^{(i)} = f(\hat{x}^{(i)})$
$V \leftarrow \{\hat{y}^{(1)}, \dots, \hat{y}^{(N)}\}$
return $T \leftarrow$ `RecTree`$(\mathcal{X}, S, V)$

GLEAMS computational complexity directly relates to the number N of Sobol points instead of the dataset size, enabling it to be computed in reasonable time even for very large datasets. However, since GLEAMS relies on a tree-like partitioning, maximum depth is unknown a priori, making hard to state its complexity. We only provide a rough complexity for the $B_j(t)$ process, which is $\mathcal{O}(Nd^2n_{\min})$. Notice how $B_j(t)$ complexity is much lower than linear models complexity of roughly $\mathcal{O}(N^3)$.

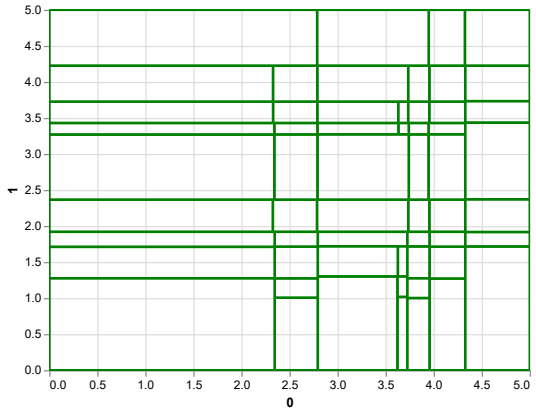


Figure 8.4: Partition of $\mathcal{X}$ produced by Algorithm 7. On each rectangular cell of $\mathcal{P}$, a linear model is fitted on the Sobol points evaluations of f.

8.1.4.0. GLEAMS Explanations

Let us now assume that we computed the global surrogate model $\hat{f}$, that is, a piecewise-linear model based on a partition of $\mathcal{X}$. At query time, each new example to explain $x^{(i)} \in \mathcal{X}$ is rooted to the corresponding hyper-rectangle R in the tree structure, and thus associated to a linear model $\hat{\beta}^R$, since the local linear model is fixed inside the cell R. From this local linear model, we can directly get different sorts of explanations.
We start with ***local attributions***: simply looking at the coefficient $\hat{\beta}_j^R$ tells us how important feature j is to predict $f(x^{(i)})$, locally. The intuition here is that f is well-approximated by $x \mapsto (\hat{\beta}^R)^\top x^{(i)}$ on the cell R, thus local variations of the model along specific axes are well-described by the coefficients $\hat{\beta}_j^R$. To put it plainly, if $\hat{\beta}_j^R \gg 0$, then we can say that feature j has a large, positive influence on predicting $f(x^{(i)})$. As explained in the introduction, these local explanations can be obtained in ***constant time***, without re-sampling and querying the model. Moreover, we have precise insights regarding the ***scale*** of these explanations, which is given by the size of R. We see this as an advantage of using GLEAMS: the portion of space on which the black-box model is locally approximated is well-identified and accessible to the user. This is not the case with methods such as LIME, where the local region depends on hyper-parameters that are not fully translatable into variable boundaries.
Another type of explanations that we can obtain in a straightforward fashion are ***global attributions***. We define the global importance of feature j as the aggregation of the local importance of feature j on all cells of the partition $\mathcal{P}$ of $\mathcal{X}$. There are several ways to aggregate those. GLEAMS outputs the weighted average of the absolute value of the local importance. More precisely, the global importance of feature j is given by

$$I_j := \frac{1}{\mathcal{V}(\mathcal{X})} \sum_{R \in \mathcal{P}} \mathcal{V}(R) \left| \hat{\beta}_j^R \right| , \tag{8.6}$$

where $\mathcal{V}(E)$ denotes the d-dimensional volume of E. The motivation for weighting by the cells' volume in Eq. (8.6) is to take into account the fact that even though feature j is very important in a tiny region of the space, if it is not on all other cells then it is not

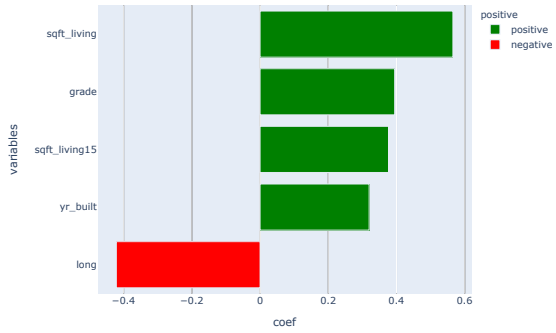


Figure 8.5: Local importance (feature attribution) for a regressor trained on the house sell dataset (see Chapter 8.2.2). We display only the top 5 features to improve readability.

globally important. We take the absolute value to take into account the possible sign variations: if feature j is positively important on many cells but negatively important on a similar number of cells, then simply taking the average would tell us that the feature is not globally important. Such ***global attributions*** provide a faithful variables ranking, based on the impact they show on f predictions.

Notice that Eq. 8.6 exploits the hyper-volume $\mathcal{V}(R)$ to compute the weighted average of the local coefficients $\hat{\beta}_j^R$. This procedure assigns different importance to each partition R solely based on its size, yielding ***true to the model*** global attributions. It is straightforward to modify the weights in Eq. 8.6 to obtain ***true to the data*** global attributions: it is just required to consider the percentage of D_{train} units falling inside the given partition R

$$I_j := \frac{1}{N} \sum_{i=1}^{N} \left| \hat{\beta}_j(x^{(i)}) \right| , \tag{8.7}$$

where $\hat{\beta}_j(x^{(i)})$ corresponds to $\hat{\beta}_j^R$ for the partition R in which the D_{train} unit $x^{(i)}$ falls in. ***True to the data*** attributions can also be obtained as an average over partitions instead of over D_{train} units, by previously computing the fraction of units in each partition R. *-This speeds up the computation, since we usually obtain a much lower number of partitions than training points-*. Equation 8.7 gives more importance to relevant partitions, namely regions with high data density, while regards low density partitions as less important. Equation 8.7 can be computed on any dataset different from D_{train} .
In the following experiments, we consider only the ***true to the model global attributions***, to be able to thoroughly inspect GLEAMS explanations quality in a controlled environment. However, GLEAMS ***true to the data global attributions*** can be very useful in business use-cases and they are straightforward and computationally cheap to compute once we already retrieved the GLEAMS global surrogate model.

Finally, another type of explanations that GLEAMS can provide are ***answers to "what if" questions***. Indeed, simply by querying the GLEAMS piecewise linear model $\hat{f}(x^{(i)})$

at different values of x_j, we can answer to the question "what would be the decision if we changed feature j by this amount?" More precisely, for a given example $x^{(i)}$ and a feature j, we can look at the evolution of $\hat{f}$ when all coordinates of the input are fixed to $x^{(i)}$, except the j-th: it suffices to travel in the cells intersecting the line of equation $x_j = \xi_j$, with ξ_j ranging in $[a_j, b_j]$ -*boundaries of the feature* X_j-, and read the coefficients $\hat{\beta}_j^R$ of the local linear models associated with the R partitions intersected by the line. These what-if explanations provide answers to both local and global changes of the variable X_j. Moreover, they can be computed quickly, since there is no sampling and further calls to f -*we just inspect the set of partitions to retrieve the relevant ones*-. This allows us to present the user with a cursor which can be moved to set the feature value and present the user with explanations in real time (see Figure 8.6).

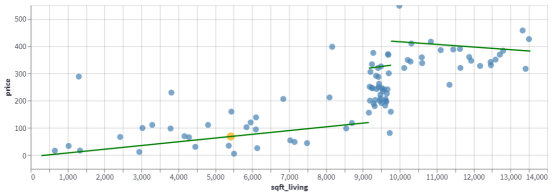


Figure 8.6: What if my house was bigger? Answering what-if questions for a regressor trained on the house sell dataset (see Chapter 8.2.2). Moving along feature `sqft_living`, for a given example (yellow dot), we can visualize specific values of f (Sobol points in blue) as well as the linear approximations used by GLEAMS (in green).

It is interesting to notice that global methods consisting of averaging local LIME explanations, such as GlocalX [101], ideally compute quantities with similar meaning to Eq. 8.6 and can be considered similar to GLEAMS under this aspect. Although, GLEAMS provides inherent fidelity scores for each single linear model ensuring that the piecewise linear approximation is faithful to the ML model. This is not always the case for LIME explanations, as explained in detail in Chapters 5,7. Another important difference consists in ***GlocalX*** being able to compute ***true to the data*** global attributions ***only*** -*since GlocalX averages feature importances computed over the entire* D_{train} *data*-, while ***GLEAMS*** provides both global ***true to the data*** and ***true to the model*** feature importances, along with the possibility of evaluating ***what-if scenarios***. The whole set of GLEAMS explanations are computed without additional overhead, provided that the only ingredient is the domain partition and the related local linear coefficients.

8.2.0. Experiments

In this Chapter, we show GLEAMS at work in different situations. We first demonstrate in Chapter 8.2.1 how the partitioning scheme is able to recover the structure of the model surface, on toy data. In Chapter 8.2.2, we show that explanations provided by GLEAMS are on par with other ***post-hoc*** methods (but without recomputing the explanations for each new example).

8.2.1.0. Toy data

As a first sanity check, we ran the partition scheme of GLEAMS on simple models and checked whether the surrogate model $\hat{f}$ was indeed close to the true model f. In order to have a ground-truth for the partition, we chose partition-based models, ***i.e.***, there exists a partition $\mathcal{P}$ of $\mathcal{X}$ such that the parameters of f are constant on each rectangular cell $R \in \mathcal{P}$. We considered a simple setting where the model is piecewise-linear with rectangular cells, a situation in which GLEAMS could, in theory, recover perfectly both the underlying partition and the coefficients of the local models. We present here two examples: (i) a discontinuous piecewise-linear model with arbitrary coefficients, and (ii) a piecewise-linear model with continuity constraints. Example (i) is a generalization of the surface produced by a Random Forest regressor, while example (ii) is reminiscent of a fully-connected, feedforward, ReLU activated neural network. Note however that for the latter, the cells of the partition are more complicated than simple rectangles, see Montufar et al. [85] (in particular Figure 8.1). In each case, GLEAMS recovers perfectly the underlying model, as demonstrated in Figure 8.7.

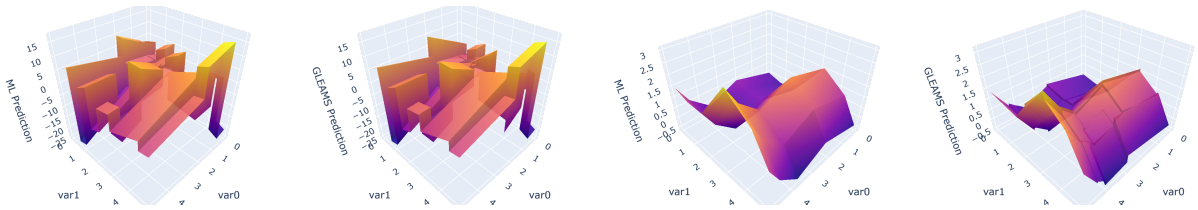


Figure 8.7: Approximating piecewise-linear models with GLEAMS surrogate model. ***Left panels:*** example (i), model on the left and GLEAMS surrogate on the right. ***Right panels:*** example (ii), model on the left and GLEAMS surrogate on the right.

It is interesting to notice that, despite having a stopping criterion on the coefficient of determination, not all leaves satisfy this criterion since the other stopping criterion is a minimal number of points per leaf. Thus it is possible to end up in situations where the R^2 on a given leaf is sub par. Nevertheless, these leaves are by construction quite small. For instance, in example (i), the average R^2 is equal to 1, and in example (ii) it is equal to .98.

8.2.2.0. Real data

We now report results on the UCI real-world data described in Chapter 3.3. Our goal here is to show that GLEAMS provides explanations whose quality is comparable to that of other attribution approaches, while providing these attributions in constant time and additional insights.
Among the different evaluation frameworks described in Chapter 4.3, we choose to compare GLEAMS to existing methodology using monotonicity and recall.
For the monotonicity, we consider two natural choices for the Uniform distribution p in Eq. (4.9). First, $p \sim \mathcal{U}([\ell_j, r_j])$, where ℓ_j (resp. r_j) are the left (resp. right) boundaries of R along feature j, where R is the cell containing $x^{(i)}$, which yields the ***local monotonicity*** metric: how well the attributions reflect the variations in prediction locally, that is,

from the point of view of GLEAMS, on R. Second, $p \sim \mathcal{U}([a_j, b_j])$, which corresponds to ***global monotonicity***: how well the attributions reflect the variations in prediction on the whole input space along feature j. Regarding the Recall metric, we report the average recall over all points of the test set unless otherwise mentioned.

Regarding the ML model applied on the three datasets, we consider only ***XGBoost*** and the ***multi-layer perceptron*** (MLP) regarding Linear Regression as a pretty simple model which does not require explanations and which should not pose any specific complication to the different explainability methods.

Methods. We compared GLEAMS to three methods, namely LIME for tabular data, SHAP, and PDP. We used default parameters, except for the number of feature combinations tested in SHAP for the recall metric, which were shrunk to 500 to limit computing time. GLEAMS results have been obtained using $N = 15$ (number of Sobol points), which guaranteed a running time in the order of 5 minutes for each of the three datasets at hand.

Results. We present our results in Table 8.1, reporting the average metrics taken on the test set. We recall that all evaluation methods are described in Chapter 4.3.
PDP generally achieves better monotonicity metrics, since it is designed precisely to obtain feature ***attributions*** related to the feature impact on the predictions, but it performs worse on the recall task. We notice that GLEAMS shows better results on ***local*** and ***global monotonicity*** compared to LIME and SHAP on the wine dataset, while GLEAMS performances start to degrade with an increased number of features. In particular we notice the poor performance on ***local monotonicity*** of the MLP model for the Parkinson dataset. The degradation is due to the higher dimensionality of the two datasets, which makes models surface more complex (MLP in particular). Increasing N would likely improve the results, enabling smaller partitions and more accurate local models. Regarding the recall metric, SHAP stands as the best in class, while GLEAMS is usually slightly worse but still comparable with LIME, and systematically better than PDP.
In general, we consider GLEAMS on par with state-of-the-art explanation methods, in terms of reliability of explanations.

To sum up, we presented GLEAMS, a new interpretability method that approximates a black-box model by recursively partitioning the input space and fitting linear models on the elements of the partition. Both local and global, true to the data and true to the model, as well as what-if explanations can then be obtained in constant time, for any new example, in sharp contrast with existing ***post-hoc*** methods such as LIME and SHAP -*which provide only specific types of explanations, and require retraining of the algorithm to explain a new instance*-.
Perhaps the main limitation of the method is the assumption that all features are con-

Table 8.1: Experimental results.

dataset	eval method	LIME		SHAP		PDP		**GLEAMS**	
		XGB	MLP	XGB	MLP	XGB	MLP	XGB	MLP
wine	local monoton	0.38	0.51	0.40	0.53	0.56	0.84	0.51	0.77
	global monoton	0.24	0.74	0.27	0.78	0.70	0.89	0.36	0.85
	recall (6 true feat)	0.89	0.80	1.0	0.87	0.50	0.50	0.77	0.75
house	local monoton	0.05	0.63	0.52	0.82	0.51	0.83	0.37	-0.19
	global monoton	0.23	0.63	0.37	0.81	0.38	0.82	0.47	0.24
	recall (10 true feat)	0.85	0.74	0.99	0.69	0.40	0.40	0.61	0.58
Parkinson	local monoton	0.10	0.26	0.47	0.55	0.28	0.49	0.50	0.01
	global monoton	0.31	0.46	0.41	0.71	0.52	0.66	0.30	0.36
	recall (10 true feat)	0.86	0.77	0.97	0.80	0.40	0.40	0.50	0.60

tinuous. As future work, we aim to extend GLEAMS to mixed features (both categorical and continuous). Another limitation of the method is the initial number of model queries needed. One possible way to reduce it would be to evaluate the local regularity of f and to sample less when the model is smooth.

Chapter 9

Explainability for Synthetic Data

Eventually, we consider also the complex scenario related to Synthetic Data.
Synthetic data is artificially generated data that is designed to closely resemble real-world data. It can be used in a variety of business scenarios, such as testing and training machine learning models, creating realistic simulations, and protecting sensitive data. There are several reasons why synthetic data is important in business scenarios:

- Data privacy: Synthetic data can be used to protect sensitive data by replacing it with artificially generated data that preserves the statistical properties of the original data, but without any of the personal or confidential information. This allows businesses to use data for testing and training purposes without risking a data breach.
- Data availability: In some cases, businesses may not have access to the data they need for testing and training purposes. Synthetic data can be used to generate the necessary data, allowing businesses to proceed with their projects without delays.
- Data diversity: Synthetic data can be used to generate a diverse range of data, including data from rare or hard-to-collect categories. This can be useful for testing and training machine learning models that need to be able to handle a wide range of input data.
- Data quality: Synthetic data can be generated to be free of errors, biases, or other issues that may be present in real-world data. This can make it easier to obtain high-quality results when testing or training machine learning models.
- Data cost: Generating synthetic data can be cheaper and faster than collecting real-world data, especially if the data needed is difficult or expensive to collect. This can save businesses time and resources.

Because of these reasons, Synthetic Data usage is becoming more and more frequent in business scenarios. In particular, there is a huge interest in using synthetic data to build complex Machine Learning models.
This occurrence adds another level of complexity in order to explain these models. In fact, we are required to understand what the model learnt from the synthetic data, but

at the same time also whether the synthetic data represents well the original data. If the second assumption does not hold, generated explanations will be useless since they do not relate to real-world scenarios.

Our goal is therefore to understand the fidelity degree of synthetic data compared to the original data used to generate them, from now on we will call it synthetic data quality. At the same time, we must strive to protect the privacy of the original data, in fact privacy and data quality usually behave as two antagonistic features.

Hereafter we provide a summary of the salient characteristics of Synthetic Data and we provide both theoretical and practical means to evaluate them. This is a fundamental step to ensure that synthetic data are reliable and resemble original data, so that explanations based on them will be meaningful.

Recall that D_{train} and D_{test} are two dataset coming from the same Data Generating Process (DGP). They usually arise by splitting the original data into two separate sets, in order to train unbiased ML models. In this situation, D_{train} is the dataset used to fit a generative model G with the capability of producing a new dataset, D_{synth}. To avoid any bias, D_{test} will act as the benchmark dataset.

Previous approaches to synthetic data evaluation are well summarized into three concepts:

- D_{synth} shall retain the statistical properties of the original data, i.e. D_{test}.
- Data Utility: Prediction models built respectively on D_{train} and D_{synth} shall be the most similar as possible. -*Model comparison is carried out on* D_{test}-
- Privacy of the D_{train} individuals and new ones (emulated by D_{test}) shall be guaranteed.

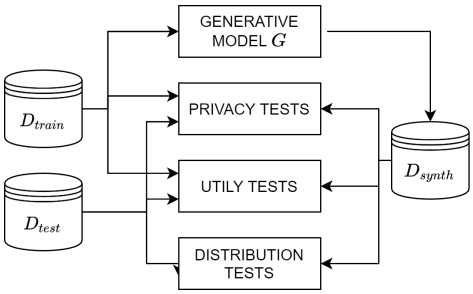


Figure 9.1: DAISYnt Pipeline

Hereafter, we review statistical similarity and data utility, while the privacy topic, has its own dedicated Chapter 9.4. Hittmeir, Ekelhart, and Mayer [53] compare the two correlation matrices and evaluate data utility comparing an aggregate performance metric, namely MAE. The same authors in [54] consider dataset similarity as well, by matching the histograms of univariate distributions. Unfortunately, the comparison relied on visualization techniques, without providing any similarity metric. In [118], the correlation

matrix comparison is performed either, while the similarity of univariate distributions is assessed using the Kolmogorov-Smirnov (KS) two-sample test. In [30] correlation is calculated by means of a particular ϕ coefficient (enabling its usage on both categorical and continuous variables). In [63] the Synthetic Ranking Agreement (SRA) is proposed to assess data utility, even if it is based on heuristic assumptions. In [1], the authors devise three separate metrics to evaluate distribution similarity employing innovative tools. Eventually, the Synthetic Data Vault (SDV) [89] provides a sandbox for open-source generative models and related quality metrics.

9.1.0. General Comparison Tests

In many business scenarios, data is solely used for descriptive statistics. The following tests check D_{synth} usefulness for these general-purpose, yet very popular, tasks.

9.1.1.0. Pairwise Correlation

Pairwise correlation is a well-known technique to measure the strength of the association between two variables. The test compares the pairwise Spearman correlation matrices $\mathbf{R}^T, \mathbf{R}^S$, respectively obtained on the D_{test} and D_{synth} datasets. Spearman retrieves a good amount of non-linear correlations, while it behaves as Pearson coefficient for linear dependence. However, it requires numerical variables. To this end, we encode categorical variables using the CatBoost encoding [21], i.e. an improved version of the Target Encoding which provides guarantees about no information leakage. The technique retains the relationship between the variable to be encoded and the target, while it may distort associations with other independent variables. To compensate for that, the Catboost Encoder is trained on the D_{test} and its mapping is applied to D_{synth}, ensuring the same distortion on both datasets, i.e. preserving their similarity.

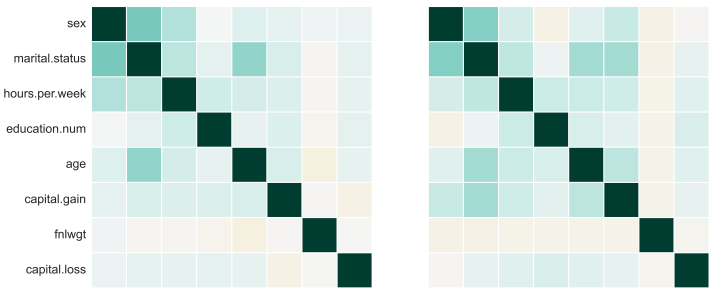


Figure 9.2: $\mathbf{R}^T, \mathbf{R}^S$ matrices for the Adult dataset

Matrix similarity is evaluated through a rescaled version of the Frobenius norm [51]:

$$d_{\text{corr}}\left(\mathbf{R}^T, \mathbf{R}^S\right) = 1 - \frac{\text{tr}\left\{\mathbf{R}^T\mathbf{R}^S\right\}}{\|\mathbf{R}^T\|_F \|\mathbf{R}^S\|_F} = 1 - \frac{< \mathbf{R}^T, \mathbf{R}^S >_F}{\|\mathbf{R}^T\|_F \|\mathbf{R}^S\|_F}$$

The Frobenius inner product concatenates the rows of the matrix and computes the

euclidean inner product between the two vectors. Intuitively, it consists of element-wise comparison of the two matrices.

9.1.2.0. Predictive Power comparison

The Information Value (IV) [121] measures the strength of the association between each variable and the target, commonly employed in Credit Scoring for feature selection. It requires categorical variables, so the continuous ones have been binned according to the deciles of their marginal distribution. We compute two IV vectors IV^T, IV^S on D_{test} and D_{synth} respectively, and measure their similarity by means of Pearson linear correlation:

$$\rho(IV^T, IV^S)$$

High values ensure that D_{synth} maintains the same variables ranking and relative distance between values, while specific IV numbers might differ.

	sex	marital.status	hours.per.week	fnlwgt	education.num	capital.loss	capital.gain	age
IV test	0.316459	1.234259	0.455335	0.002725	0.690320	0.178502	0.583700	1.020343
IV synth	0.176601	1.124148	0.296753	0.000768	0.388314	0.268912	0.526058	0.903863

Figure 9.3: IV^T, IV^S vectors for the Adult dataset

9.2.0. Distributions Comparison Tests

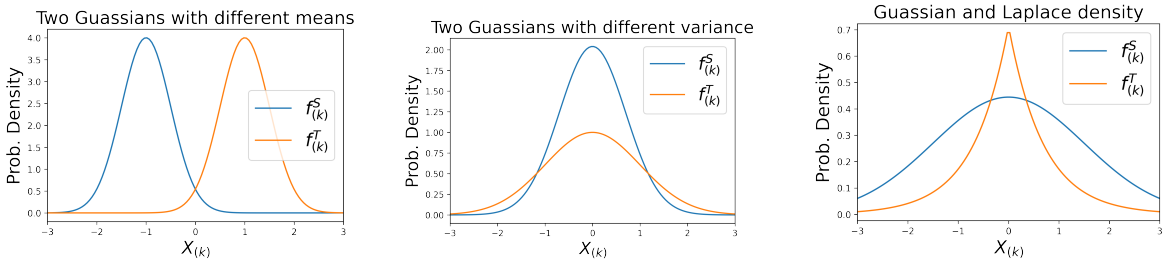
The most powerful tool to describe a dataset is the multivariate distribution: it encodes all the variables information, such as moments, single feature marginal distribution and multivariate distribution for groups of features, pointing out correlations and interactions. Having considered estimating the D_{test}, D_{synth} (pdf) and subsequently comparing them, we recognize this is not a viable solution. In fact, parametric estimation is not easily generalizable since it requires domain knowledge for the distribution choice, while non-parametric methods do not provide pdf formulas for the comparison. Therefore, we opted for discrepancy tests between the two pdf, without explicitly estimating them.
Moreover, since we deal with finite datasets, multivariate tests may fail. Thus, we detach univariate and multivariate distribution comparison. This ensures accurate results on the low-dimensional tests while it provides a theoretically solid framework to perform high-dimensional testing, even being aware they could fail due to a huge number of features.

9.2.1.0. Univariate Distributions

There are no proper techniques working well for both categorical and continuous features. Convenience solutions entail converting continuous variables into categorical, through binning methods, at the cost of losing information (we lose granularity coercing continuous variables with many different values into a fixed number of bins). Conversely, it is difficult to convert categorical features to continuous ones without distorting the multivariate distribution. We keep the two comparisons separated, in order to employ more powerful tests.

Univariate Distribution - Continuous variables Consider the generic continuous variable $X_{(k)}$: $X^T_{(k)}$ ($X_{(k)}$ values in D_{test}) are drawn from the $f^T_{(k)}$ marginal distribution, while $X^S_{(k)}$ ($X_{(k)}$ values in D_{synth}) stem from $f^S_{(k)}$. The test goal is to assess whether the null hypothesis $\mathcal{H}_0 : f^T_{(k)} = f^S_{(k)}$ holds.

Distributions are fully characterized by the collection of their moments (eg. mean, variance, skewness, kurtosis etc), hence an intuitive approach is to search for differences in the $X^T_{(k)}, X^S_{(k)}$ moments. Regardless of which moment is involved, it is possible to distinguish the two distributions by choosing a proper q function and evaluating its expectation. Two equal distributions shall obtain similar expected values for each valid q. As an example, difference in variance can be grasped using $q(x) = x^2$.



(a) Difference in Mean (b) Difference in Variance (c) Higher-order moments diff

Figure 9.4: Examples of different distributions

Maximum Mean Discrepancy (MMD) [39] looks for the q maximizing the difference between the two expectations:

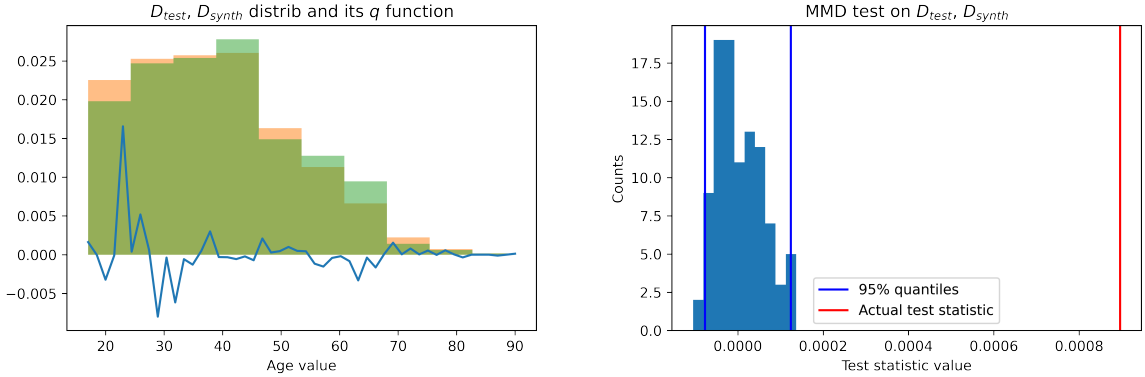
$$\text{MMD}[\mathcal{Q}, X^T_{(k)}, X^S_{(k)}] := \sup_{q \in \mathcal{Q}} \left(\frac{1}{m} \sum_{i=1}^{m} q\left(x^T_{(k),i}\right) - \frac{1}{n} \sum_{j=1}^{n} q\left(x^S_{(k),j}\right) \right) \tag{9.1}$$

Here m, n respectively stand for the D_{test}, D_{synth} sample size, while expectations have been replaced by the sample means. Under mild regularity conditions ($\mathcal{Q}$ contains only continuous, bounded, smooth functions), MMD is guaranteed to be different from 0 only when $f^T_{(k)}, f^S_{(k)}$ are truly different. However, the superior operator requires screening all the valid $q \in \mathcal{Q}$, making Equation 9.1 impractical to compute. An efficient MMD formulation involves kernel functions:

$$\left[\frac{1}{m^2} \sum_{i,j=1}^{m} k\left(x^T_{(k),i}, x^T_{(k),j}\right) - \frac{2}{mn} \sum_{i,j=1}^{m,n} k\left(x^T_{(k),i}, x^S_{(k),j}\right) + \frac{1}{n^2} \sum_{i,j=1}^{n} k\left(x^S_{(k),i}, x^S_{(k),j}\right) \right]^{\frac{1}{2}}$$

The usual choice for $k(\cdot, \cdot)$ is the Gaussian kernel, thanks to its ability to screen infinitely large function spaces. In our implementation of the test, we take advantage of the heuristic provided in [110] to choose the proper value for σ (the only hyper-parameter for the Gaussian kernel).
Eventually, we estimate the distribution of MMD under the Null Hypothesis $\mathcal{H}_0$, using a permutation test [110]. It essentially consists of randomly partitioning the data $D_{test} \cup$

D_{synth} into D'_{test} and D'_{synth} (under $\mathcal{H}_0$ the two samples come from the same distribution) and computing the MMD. Repeating it many times, we obtain a numerical approximation of its distribution. With it, we calculate the acceptance interval (considering a 5% type I error), through which we establish whether the variable $X_{(k)}$ passed the univariate similarity test.



(a) D_{test}, D_{synth} distribution histograms

(b) Confidence Interval for the MMD test

Figure 9.5: MMD Test on the Age variable. Here the test failed, meaning that Age distribution is different in D_{test} and D_{synth}

While MMD is one of the most powerful tests for distribution similarity, it does not scale well for large datasets (due to kernels). In DAISYnt the test runs on D_{test} and D_{synth} sub-samples, whose size can be set by the user to match its computational and hardware requirements. An additional option for extremely large datasets, is to bin continuous variables using deciles and perform the faster categorical distribution testing, at the cost of a coarser result.

Univariate Distribution - Categorical variables The MMD based test is no use for categorical variables, therefore DAISYnt incorporates a classic Chi-Square two-sample test. The null Hypothesis $\mathcal{H}_0$ "no significant difference in the $X_{(k)}$ class frequencies between D_{test} and D_{synth}" has formula:

$$\sum_{k=1}^{C} \frac{\left(f^T_{(k),c} - f^S_{(k),c}\right)^2}{f^T_{(k),c}} \sim \chi^2_{(n-1)\times(C-1)}$$

where c shuffles through the different $X_{(k)}$ classes, $f^T_{(k)}, f^S_{(k)}$ are the class frequencies in D_{test}, D_{synth}. Under $\mathcal{H}_0$, the test exhibits a χ^2 distribution with degrees of freedom $df = (n-1) \times (C-1)$, which allows controlling the type I error (DAISYnt default is 5%). Eventually, it is important to check whether the assumptions are met, in particular the test is meaningful only if at least 80% of the classes have an expected frequency (in this case $f^T_{(k)}$) greater than 5 and no class has 0 frequency in $f^S_{(k)}$ [25]. Following these prescriptions, DAISYnt do not test variables not complying with such criteria.

To obtain a unified metric, DAISYnt applies the proper test to categorical and continuous variables separately and calculates the fraction of accepted trials.

9.2.2.0. Multivariate Distributions

As already stated, high dimensionality and mixed data are notable challenges for two-sample distribution tests. To this end, feature selection is employed and categorical and continuous variables are considered separately. As a minor limitation, DAISYnt cannot discover any difference in distribution for groups of mixed variables.

Multivariate Distribution - Continuous Variables Based on empirical experiments, we deduce that MMD greatly suffers the curse of dimensionality. Hence, we restrict the similarity test to the subset of meaningful variables only, namely the relevant ones for a given prediction task. Boruta [72] is the method of choice, thanks to its stability, reliability and reasonable computational time.

Boruta improves the Tree-based feature importance, known to be particularly unreliable in presence of correlation and interactions among variables [108]. The method creates "shadow features" (variables with no relationship with the target variable) and iteratively generates Random Forest models, computing confidence intervals for the feature importance scores. Variables with significantly higher feature importance than any shadow feature are the relevant ones. Repeated trials ensure statistical stability of the procedure, while the choice of Random Forest models, which do not require any fine-tuning, guarantees an acceptable computation time.

DAISYnt relies on Boruta to extract the set of numerical meaningful features and runs the MMD test on it. The test is much more likely to fail whenever one of the important variables had already failed the univariate test. Moreover, even if all the single variables passed the univariate test, the multivariate one may fail due to differences in correlations and interactions. Substantially, the multivariate comparison is more challenging but, in case of acceptance, gives strong evidence of distribution similarity.

Multivariate Distribution - Categorical Variables We propose to extend the Chi-Square test to groups of categorical variables by considering a new feature, whose classes are the cartesian product of the group variables' classes, and frequency given by the contingency table. The test is carried out on the new feature. Notice that the more variables in the group, the more likely the test assumptions are not met.

DAISYnt considers the power set of the categorical variables. Starting from the 2-variables groups, it checks whether the test assumptions are fulfilled. If so, the Chi-Square test is carried out and the acceptance/rejection is recorded, otherwise the group is discarded. Bigger groups are tested only if all the sub-groups met the assumptions (otherwise the super-group would fail them either). The fraction of accepted tests is the DAISYnt similarity metric for multivariate categorical distributions.

9.2.3.0. Discriminator Model

Eventually, we propose to aggregate D_{train}, D_{test} and D_{synth} together, label each tuple as original/synthetic, and train a discriminator model to predict the label. Both D_{train}, D_{test} are used to ensure test robustness, for small D_{test} dataset. Even if this test is not explicitly concerned with distributions, discriminators obtain good performance by implicitly learning distributional differences between the two datasets.

DAISYnt exploits a Gradient Boosting d_{xgb} and a shallow Neural Network d_{nn} as discriminators (more on model specifics in the Data Utility Chapter 9.3), and evaluates their performance using Gini Index. Since we strive for synthetic data to be not distinguishable from the real ones, the discriminator test metric is rescaled as follows:

$$1 - \frac{Gini(d_{xgb}) + Gini(d_{nn})}{2}$$

9.3.0. Data Utility Tests

Synthetic data have many different applications, not least to replace real data in prediction tasks. Conceptually, two datasets stemming from the same multivariate distribution shall yield the same insights, but this is not always the case when advanced models are involved.
To ensure synthetic data maintain data utility, DAISYnt trains prediction models respectively on D_{train} and D_{synth} and devises three tests with an increasing level of detail.

The models employed are Gradient Boosting (Xgboost [16] implementation) and Neural Network. Xgboost is trained with shallow decision trees of 4 splits. The Neural Network consists of a single hidden layer of 256 neurons, with Relu activation function. Early stopping is used to avoid overfitting. Both frameworks have been trained twice on the D_{train} and D_{synth} datasets, obtaining four models GB^S, GB^T, NN^S, NN^T. DAISYnt compares the two frameworks separately, using D_{test} as benchmark.

9.3.1.0. Aggregate Prediction Comparison

Since DAISYnt mainly focus on binary classification tasks, it computes the AUC difference on the D_{test} predictions of the four models, considered in pairs. Assuming that models trained on D_{synth} cannot approximate the true underlying DGP better than the D_{train} ones (well-grounded assumption in practice), the metric is rescaled as follows:

$$1 - \frac{AUC(GB^T(D_{test})) - AUC(GB^S(D_{test}) + AUC(NN^T(D_{test})) - AUC(NN^S(D_{test})}{2}$$

9.3.2.0. Single Prediction Comparison

Woefully we acknowledge that AUC, due to its aggregate nature, does not guarantee the similarity of single predictions. Conversely, obtaining the same prediction probability

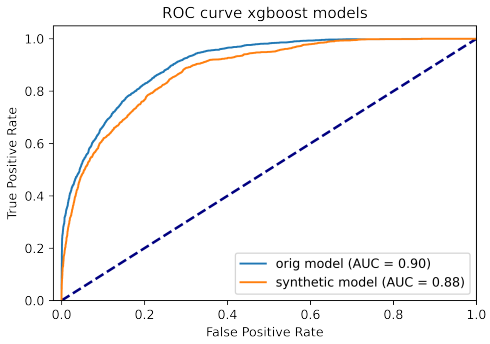


Figure 9.6: ROC Curve comparison of GB^S, GB^T models, and relative AUC

rankings (on the D_{test} individuals) is important to produce the same insights.

To this end, we define the D_{test} target variable as Y and the models predictions as $\hat{Y}^S_{GB}, \hat{Y}^T_{GB}, \hat{Y}^S_{NN}, \hat{Y}^T_{NN}$. DAISYnt compares prediction vectors using the Cosine Similarity, which takes into account solely vector direction (corresponding to the ranking concept for predictions). Cosine similarity is evaluated on the prediction vectors, for Gradient Boosting and Neural Network separately. Results are then aggregated into a final test metric:

$$\frac{CS(\hat{Y}^S_{GB}, \hat{Y}^T_{GB}) + CS(\hat{Y}^S_{NN}, \hat{Y}^T_{NN})}{2}$$

9.3.3.0. Compare model internals

The most refined and challenging comparison step is to check whether the model internals are the same. In a Neural Network, data flows through the network and gets transformed according to the neurons weights. We define layer activations as the values obtained by passing a data matrix through the network up to the chosen layer. The objective is to compare the activations of the NN^T and NN^S hidden layers, when passing D_{test} as the data matrix. However, Neural Networks employ a huge number of parameters that frequently cause an over-representation, i.e. there are possibly infinite ways of achieving the same prediction, via different intermediate layer values. We cannot compare activations directly: they would always be different. Rather, we consider two activations to be equal when they are isotropic scaling or orthogonally invariant.

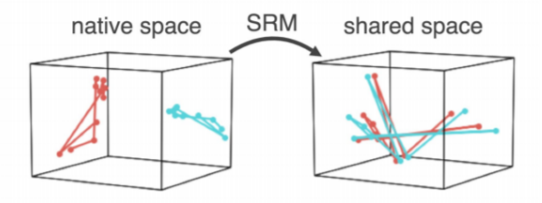


Figure 9.7: Toy example of transformations applied to activations matrices [69]

The HSIC quantity [40] captures orthogonal transformations but cannot handle the Isotropic Scaling invariance, which can be achieved by rescaling it into the Centered Kernel Alignment (CKA) [69]. Both CKA and HSIC employ kernels to achieve the required transformations. Popular options are the linear and gaussian kernels, where the difference consists of a trade-off between flexibility and computation time.

DAISYnt default consists in linear CKA between the NN^T and NN^S activations matrices, since the linear kernel usually achieves very similar results to the gaussian one [69].

9.4.0. Privacy Tests

Since there exists no privacy regulation yet on synthetic data, we consider the taxonomy of privacy risks related to personal data, issued in the Article 29 Working Party Opinion 05/2014. Only the Linkability risk definition is slightly modified, to emphasize that building a generative model implies privacy obligations towards the D_{train} individuals. The privacy dangers are grouped as:

- ***Singling out risk***: The attacker single out an individual, using D_{synth}.
- ***Linkability risk***: The attacker learns whether a given D_{train} record has been used to train the generative model.
- ***Inference risk***: The attacker infers the value of a sensitive attribute for new records, using the relationships contained in the D_{synth}.

Hereafter, the focus is on black-box privacy attacks (the attacker has access to synthetic data only) and how to defend against them, rather than on formal privacy frameworks such as k-anonymity, l-diversity or differential privacy [4], which are difficult to evaluate post-hoc. Membership Inference attacks usually exploit an overfitted generative model which creates D_{synth} records too close or too similar to the D_{train} ones. Closeness is used in [52], while the similarity in [106], to retrieve which records belong to D_{train}. A different attack can be devised using the same procedure as [64], i.e. training a model to predict a D_{synth} sensitive variable and exploit it to infer confidential information on new records. In the following we will present tests useful either to ensure that privacy is not at risk or to spot potential vulnerabilities.

9.4.1.0. Singling Out Tests

Cloned rows test The first test checks the presence of equal records in D_{train}, D_{synth} and measures their percentage. Cloned rows can be removed from the synthetic data, at the cost of having fewer data and modifying the statistical properties of the dataset.

Close rows test The second test looks for very similar rows. DAISYnt converts continuous variables into categorical through binning, on both D_{train} and D_{synth}, and use the new categorical versions to calculate the Hamming distance for each original-synthetic pair of records. Pairs with Hamming distance < 2 are considered close. The test metric is the fraction of D_{synth} records which are close to no D_{train} records.

	age	fnlwgt	education.num	marital.status	sex	capital.gain	capital.loss	hours.per.week	income
D_{train}	50	30682	14	1	0	3273	0	50	0
D_{synth}	50	30682	14	1	0	3273	0	50	0
D_{synth}	50	30682	15	1	0	3272	0	50	0

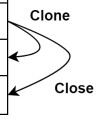


Figure 9.8: Cloned and close rows between D_{train} and D_{synth}

9.4.2.0. Linkability Tests

An overfitted G model may potentially leak information about the dataset used for training. In practice, an attacker may re-identify the D_{train} records starting from D_{synth}, if she has access to a database containing some of the D_{train} individuals.

Linkability distance test The assumption here is that an overfitted G model shall produce D_{synth} units much closer to the D_{train} ones than to D_{test} [52], i.e. choosing a fixed tiny size ϵ, the ϵ-neighbourhood of a generic D_{train} individual $U_\epsilon(x \in D_{train})$ will presumably contain more $x' \in D_{synth}$ than $U_\epsilon(x \in D_{test})$. Since distance notions suffer the curse of dimensionality, we perform Factor Analysis of Mixed Data (FAMD) [100] to retrieve a restricted set of orthogonal variables. On the new coordinates system, we compute the euclidean distance matrix between $D_{train} \cup D_{test}$ and D_{synth}. The "median heuristic" [52] supports the choice of ϵ. Per each $x \in D_{train} \cup D_{test}$ we count the fraction of D_{synth} units belonging to its neighbourhood, and use it as a ranking measure.

$$rank(x) = \frac{1}{n_{synth}} \sum_{i=1}^{n_{synth}} \mathbf{1}\left(x' \in U_\epsilon(x)\right) \qquad \text{where } x' \in D_{synth}$$

where $\mathbf{1}$ is the indicator function.
Under the initially stated assumption, D_{train} data should have higher $rank$ values, i.e. the $rank$ variable should discriminate well between D_{train} and D_{test} data. Its discrimination power is measured through AUC, obtaining the test metric:

$$1 - AUC_{rank}$$

From a different standpoint, we create a classification model exploiting the closeness of synthetic data to classify the units as D_{train} or D_{test}.

Linkability ML test Information contained in D_{synth} should be more valuable for re-estimating the D_{train} data rather than D_{test}, when G is an overfitted model [106]. To test such assumption, we build a Neural Network on the D_{synth} target variable and employ the model to obtain target prediction $\hat{y}|x$ on the x units of D_{train} and D_{test}. Considering a classification task, the cross-entropy error $Err(\hat{y}|x)$ per each single prediction is computed (yet the generality of the framework allows for any other error metric to be used).
As before, we test the the Err variable discrimination power, i.e. how well it separates D_{train} from D_{test} units. Considering the worst-case scenario, the threshold τ is chosen to maximize the proportion differences of D_{train}, D_{test} over and below τ (an attacker usually does not have enough information to choose the best τ). The test metric is computed as

follows:

$$1 - \left(\frac{\sum_{x \in D_{train}} \mathbf{1}\{Err(\hat{y}|x) < \tau\}}{n_{train}} - \frac{\sum_{x \in D_{test}} \mathbf{1}\{Err(\hat{y}|x) < \tau\}}{n_{test}} \right)$$

9.4.3.0. Inference Risk test

We assume that an attacker has access to a set of public variables X_{pub}, but only an aggregate knowledge (average) of a sensitive variable y_{sens}, i.e. $\bar{y}_{sens}$.
We measure the y_{sens} increased disclosure the attacker gains, by obtaining D_{synth}. The attacker may build a prediction model $f : \mathcal{X}_{pub} \rightarrow \mathcal{Y}_{sens}$ to predict the sensitive attribute on any individual, rather than predicting $\bar{y}_{sens}$.

Recall that R^2 metric calculates the prediction improvement using the f model, against the basic average $\bar{y}_{sens}$ prediction. We employ a slightly modified R^2 version to measure the inference risk on the D_{train} units:

$$\frac{\sum_{i=1}^{n_{train}} \left(y_i - f(X_{pubs,i})\right)^2}{\sum_{i=1}^{n_{train}} \left(y_i - \bar{y}_{sens}\right)^2}$$

Such test is valuable for the data owner to understand the dangers of sharing D_{synth} and decide how to protect against them.

9.5.0. Applications

In order to test DAISYnt consistency and to show the relevant insights it produces, we consider the Credit-Scoring dataset of Chapter 3.1. The scope of the application is to create a reliable synthetic replica to be used for data sharing with model development purposes. Given the highly regulated domain, we need to be able to provide explanations for the resulting models. In order to do so, we need to prove the Synthetic Data reliability first, to then allow ML model training and subsequently to explanations. In this example we are going to assess Synthetic Data Validity only. In a real business scenario, the practitioner can then choose the preferred explanation method and use it to produce the required explanations.

We compare four different generative models, trained to produce D_{synth} datasets of the same D_{train} size, and DAISYnt is used to determine the best replica.

Different open-source [89] model implementations have been employed: i) Gaussian Copula (GC), ii) Conditional Tabular GAN (CTGAN), iii) CopulaGAN, iv) Tabular Variational Auto-Encoders (TVAE). Out-of-the-box SDV models were used with no fine-tuning of the hyper-parameters. Concerning the test suite, DAISYnt package[1] requires only few essential parameters, namely the $D_{train}, D_{test}, D_{synth}$ datasets, knowledge of the categorical features and the target variable. These information are used for a shallow preprocessing step, in which DAISYnt takes care of missing and special values (if specified). On categorical variables they are handled using ad-hoc classes, while on continuous variables

[1] https://pypi.org/project/daisynt/

Table 9.1: DAISYnt results on the four D_{synth} datasets models

Test	Group	Detail	GC	CTGAN	CopGAN	TVAE
Correlations	General	basic	0.93	0.96	0.97	0.94
Predictive Power	General	basic	0	0.97	0.92	0.38
Uni Distrib (bins)	Distrib	basic	0.81	0.94	0.88	0.94
Uni Distrib (MMD)		in-depth	0.13	0.25	0.13	0.25
Multi-Cat Distrib	Distrib	basic	0.99	1	0.99	1
Multi-Cont Distrib	Distrib	in-depth	0	0	0	0
Discriminator	Distrib	in-depth	0.01	0.07	0.08	0.05
Aggregate Preds	Utility	basic	0.68	0.93	0.92	0.77
Single Preds	Utility	in-depth	0.02	0.46	0.42	0.12
Model Internals	Utility	in-depth	0.70	0.79	0.66	0.61
Cloned Rows	Privacy	basic	1	0.99	0.99	0.99
Close Rows	Privacy	basic	1	0.99	0.99	0.99
Linkability Dist	Privacy	basic	0.95	1	0.98	1
Linkability ML	Privacy	basic	1	0.99	1	0.99

they are treated through mean imputation and a flag indicating which records held the missing/special fields. The target variable is instead required for data utility. Additionally, we may specify the list of tests to perform, by default DAISYnt runs the entire set.

DAISYnt results are summarised in Table 9.1. All the tests described in the methodological sections have been performed, apart from the Inference Risk test. The latter is especially valuable to understand the privacy implications of sharing synthetic data, although it requires knowing which features are available to the adversary to carry out an Inference attack.
General purpose tests guarantee same correlation and predictive power patterns, proving that the datasets can be used for descriptive analysis. Concerning the distribution related tests, we notice satisfactory results for the univariate distributions (with binned variables) and multivariate categorical distributions. However, more in-depth tests, such as the MMD univariate, continuous multivariate distributions and the discriminator test, show quite poor values. Since we tested just the vanilla implementations, distribution discrepancies were expected. However, these results suggest the replicas cannot be employed for advanced statistical analyses, such as rebalancing good-bad payers [12]. About data utility, aggregate predictions testify similar AUC values, but this is not enough to enable model development on the synthetic data. In fact, the more in-depth single prediction test shows that models on D_{train} and D_{synth} achieve substantially different insights. DAISYnt has been especially valuable in determining whether the generated datasets are reliable enough to proceed further. Given the negative answer, the practitioner would be required to either fine-tune the SDV generative models to obtain better performances or to test different and more advanced generative models. Eventually, high privacy metrics

guarantee that privacy is preserved and the models do not overfit.
From a generative models standpoint, we notice that GAN models perform visibly better with respect to Gaussian Copula and TVAE, especially in terms of data utility. The generated datasets may be used for data sharing purposes, thanks to good privacy results. For model development purposes.

To sum up, we provide a taxonomy of the important aspects related to Synthetic Data and powerful tests to assess them. The DAISYnt test suite is an easy-to-use python package, characterized by modularity and flexibility. The tests are grouped by different properties and level of detail. Based on the data generation purpose, we may choose the most appropriate tests to run. In fact, different use-cases require different quality levels of each concept. As an example, data sharing use-cases usually require high privacy levels, model development needs good data utility, while data augmentation compels strong distribution similarity. In this regard, DAISYnt is particularly versatile and business oriented.

Future directions entail tests refinement and extending DAISYnt to regression and multiclass classification prediction tasks, maintaining the applied use-cases focus and the current ease of use.

Chapter 10

Conclusions

In this thesis we delved into the realm of Explanations for black-box and complex models. We specifically focused on models trained on Tabular Data, posing specific emphasis on the Credit Scoring domain.

The techniques described herein are widely applicable to any kind of prediction model built on Tabular Data stemming from any domain. In fact, we specifically deal with model-agnostic post-hoc techniques. These are essentially separate tools to be used on top of the ML algorithm, making them especially suitable in business processes involving already trained or even outdated models.

Among the well-known Explanation frameworks, our preference is towards Surrogate models, since they provide both: ***feature attributions*** and the chance to test specific ***what-if scenarios***. We recognize that Local Surrogates achieve better explanations, by means of clarity and reliability, compared to the Global ones.
For this reason we delve specifically into the LIME framework, considered the predecessor of most of the new Local surrogate techniques. We provide a thorough theoretical description, along with practical insights on each step of the algorithm, proposing improvements regarding the choice of the local explainable model. We describe the weakpoints of the method as well, giving particular attention to the instability issue and the complex task of choosing the proper local size of the explanation. For both of them, we propose state-of-the-art solutions, namely a pair of ***statistical stability indices*** and the ***OptiLIME*** automated policy.

The Stability Indices allow the user to spot instability issues in a trained LIME explanation. They are especially useful to certify reliability of single explanations, but also as a quantitive benchmark to rely on for improving unstable explanations.
Building on that, the OptiLIME policy finds the best local neighbourhood size by leveraging the trade-off between stability and adherence of the explanations.

The further step consisted in blending together the local and global explanation frameworks, maintaining the best of the two worlds. This is embodied by ***GLEAMS*** (Global and Local ExplAnations through Model Space partitioning), an innovative algorithm par-

titioning the space in a Decision Tree fashion and training a Linear model in each partition. The result is a piecewise linear model which provides local explanations with the same level of detail as LIME, but with the added bonus of knowing exactly the local boundaries, i.e. how far we can push a what-if scenario and still consider the local explanation valid. In addition, GLEAMS provides a global surrogate model, which allows for both ***global feature attributions*** and ***extended what-if scenarios***. GLEAMS requires to be trained only once, to then provide explanations for any given unit ***in constant time***.

Eventually, we consider the rising topic of Synthetic Data and its implications on Machine Learning and related Explanations. In particular, Synthetic Data are considered the new fuel for Machine Learning models: more and more companies are switching to synthetic data for a variety of reasons, one above all privacy. From the Explanations viewpoint, it is important to be aware that using synthetic data brings an additional complexity and requires us to ensure that the generation step is faithful as well. In order to do so, we propose a taxonomy of the four main concepts representing ***synthetic data goodness***. We develop specific tests to validate each concept, incorporated in ***DAISYnt***: a comprehensive, fast and easy-to-use suite of tests. Moreover, DAISYnt is broad in scope, allowing us to assess synthetic datasets suitability for a variety of different business scenarios.

List of Figures

List of Tables

Bibliography

[1] Ahmed M. Alaa et al. "How Faithful Is Your Synthetic Data? Sample-Level Metrics for Evaluating and Auditing Generative Models". 2021. arXiv: 2102.08921.

[2] David Alvarez-Melis and Tommi S. Jaakkola. "On the Robustness of Interpretability Methods". June 20, 2018. arXiv: 1806.08049 [cs, stat]. URL: http://arxiv.org/abs/1806.08049 (visited on 03/10/2019).

[3] Sebastian Bach et al. "On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation". In: ***PloS one*** 10.7 (2015), e0130140.

[4] Steven M. Bellovin, Preetam K. Dutta, and Nathan Reitinger. "Privacy and Synthetic Datasets". In: ***Stan. Tech. L. Rev.*** 22 (2019), p. 1.

[5] Kevin Beyer et al. "When Is "Nearest Neighbor" Meaningful?" In: International Conference on Database Theory. Springer, 1999, pp. 217–235.

[6] Patrick Billingsley. ***Probability and Measure***. John Wiley & Sons, 2008. ISBN: 81-265-1771-9.

[7] Robert Brame et al. "Testing for the Equality of Maximum-Likelihood Regression Coefficients between Two Independent Equations". In: ***Journal of Quantitative Criminology*** 14.3 (1998), pp. 245–261. ISSN: 0748-4518.

[8] Steven Bramhall et al. "Qlime-a Quadratic Local Interpretable Model-Agnostic Explanation Approach". In: ***SMU Data Science Review*** 3.1 (2020), p. 4.

[9] Leo Breiman. "Random Forests". In: ***Machine learning*** 45.1 (2001), pp. 5–32.

[10] Leo Breiman et al. "Classification and Regression Trees. Wadsworth & Brooks". In: ***Cole Statistics/Probability Series*** (1984).

[11] Andrew Brock, Jeff Donahue, and Karen Simonyan. "Large Scale GAN Training for High Fidelity Natural Image Synthesis". In: ***International Conference on Learning Representations***. 2018.

[12] Giuseppe Cascarino, Mirko Moscatelli, and Fabio Parlapiano. ***Explainable Artificial Intelligence: Interpreting Default Forecasting Models Based on Machine Learning***. Bank of Italy, Economic Research and International Relations Area, 2022.

[13] Sara Castellanos. "Fake It to Make It: Companies Beef Up AI Models With Synthetic Data". In: ***Wall Street Journal. WSJ Pro*** (July 23, 2021). ISSN: 0099-9660. URL: https://www.wsj.com/articles/fake-it-to-make-it-companies-beef-up-ai-models-with-synthetic-data-11627032601 (visited on 12/16/2022).

[14] Kin-Yee Chan and Wei-Yin Loh. "LOTUS: An Algorithm for Building Accurate and Comprehensible Logistic Regression Trees". In: ***Journal of Computational and Graphical Statistics*** 13.4 (2004), pp. 826–852.

[15] Hugh Chen et al. "True to the Model or True to the Data?" 2020. arXiv: 2006.16234.

[16] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: ***Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16***. The 22nd ACM SIGKDD International Conference. San Francisco, California, USA: ACM Press, 2016, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. URL: http://dl.acm.org/citation.cfm?doid=2939672.2939785 (visited on 08/02/2019).

[17] I. Glenn Cohen and Michelle M. Mello. "HIPAA and Protecting Health Information in the 21st Century". In: ***Jama*** 320.3 (2018), pp. 231–232.

[18] Paulo Cortez et al. "Modeling Wine Preferences by Data Mining from Physicochemical Properties". In: ***Decision support systems*** 47.4 (2009), pp. 547–553.

[19] Christine S. Cox. ***Plan and Operation of the NHANES I Epidemiologic Followup Study, 1987***. 27. US Department of Health and Human Services, Public Health Service, Centers . . ., 1992.

[20] Mark Craven and Jude W. Shavlik. "Extracting Tree-Structured Representations of Trained Networks". In: ***Advances in neural information processing systems*** (1996), pp. 24–30.

[21] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. "CatBoost: Gradient Boosting with Categorical Features Support". 2018. arXiv: 1810.11363.

[22] Finale Doshi-Velez and Been Kim. "Towards a Rigorous Science of Interpretable Machine Learning". 2017. arXiv: 1702.08608.

[23] European Banking Authority EBA. "Report on Big Data and Advanced Analytics". In: (2020). URL: https://eba.europa.eu/eba-report-identifies-key-challenges-roll-out-big-data-and-advanced-analytics.

[24] Jing Fang and Michael H. Alderman. "Serum Uric Acid and Cardiovascular Mortality: The NHANES I Epidemiologic Follow-up Study, 1971-1992". In: ***Jama*** 283.18 (2000), pp. 2404–2410.

[25] Murray J. Fisher, Andrea P. Marshall, and Marion Mitchell. "Testing Differences in Proportions". In: ***Australian Critical Care*** 24.2 (2011), pp. 133–138.

[26] Future of Life Institute FLI. ***The Artificial Intelligence Act***. The Artificial Intelligence Act. Sept. 7, 2021. URL: https://artificialintelligenceact.eu/ (visited on 12/09/2022).

[27] Alex A. Freitas, Daniela C. Wieser, and Rolf Apweiler. "On the Importance of Comprehensible Classification Models for Protein Function Prediction". In: ***IEEE/ACM Transactions on Computational Biology and Bioinformatics*** 7.1 (2008), pp. 172–182.

[28] Jerome H. Friedman. "Greedy Function Approximation: A Gradient Boosting Machine". In: ***Annals of statistics*** (2001), pp. 1189–1232.

[29] Jerome H. Friedman and Bogdan E. Popescu. "Predictive Learning via Rule Ensembles". In: ***The Annals of Applied Statistics*** 2.3 (2008), pp. 916–954.

[30] Andrea Galloni, Imre Lendák, and Tomáš Horváth. "A Novel Evaluation Metric for Synthetic Data Generation". In: ***International Conference on Intelligent Data Engineering and Automated Learning***. Springer, 2020, pp. 25–34.

[31] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. "A Neural Algorithm of Artistic Style". 2015. arXiv: 1508.06576.

[32] Robert D. Gibbons et al. "The CAD-MDD: A Computerized Adaptive Diagnostic Screening Tool for Depression". In: ***The Journal of clinical psychiatry*** 74.7 (2013), p. 669.

[33] Toriano Gilbert. "Family Educational Rights and Privacy Act (FERPA)". In: (2007).

[34] Alex Goldstein et al. "Peeking inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation". In: ***Journal of Computational and Graphical Statistics*** 24 (2015), pp. 44–65.

[35] Ian Goodfellow et al. "Generative Adversarial Nets". In: ***Advances in neural information processing systems*** 27 (2014).

[36] Alicja Gosiewska and Przemyslaw Biecek. "IBreakDown: Uncertainty of Model Explanations for Non-Additive Predictive Models". 2019. arXiv: 1903.11420.

[37] William H Greene. ***Econometric Analysis***. Pearson Education India, 2003. ISBN: 81-7758-684-X.

[38] Brandon M. Greenwell, Bradley C. Boehmke, and Andrew J. McCarthy. "A Simple and Effective Model-Based Variable Importance Measure". 2018. arXiv: 1805.04755.

[39] Arthur Gretton et al. "A Kernel Method for the Two-Sample-Problem". In: ***Advances in neural information processing systems*** 19 (2006), pp. 513–520.

[40] Arthur Gretton et al. "Measuring Statistical Dependence with Hilbert-Schmidt Norms". In: ***International Conference on Algorithmic Learning Theory***. Springer, 2005, pp. 63–77.

[41] Leo Grinsztajn, Edouard Oyallon, and Gael Varoquaux. "Why Do Tree-Based Models Still Outperform Deep Learning on Typical Tabular Data?" In: ***Thirty-Sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track***. 2022.

[42] Riccardo Guidotti. "Evaluating Local Explanation Methods on Ground Truth". In: ***Artificial Intelligence*** 291 (2021), p. 103428.

[43] Riccardo Guidotti and Salvatore Ruggieri. “On the Stability of Interpretable Models”. In: ***2019 International Joint Conference on Neural Networks (IJCNN)***. IEEE, 2019, pp. 1–8. ISBN: 1-72811-985-5.

[44] Riccardo Guidotti et al. “A Survey of Methods for Explaining Black Box Models”. In: ***ACM computing surveys (CSUR)*** 51.5 (2018), p. 93.

[45] Riccardo Guidotti et al. “Local Rule-Based Explanations of Black Box Decision Systems”. May 28, 2018. arXiv: 1805.10820 [cs]. URL: http://arxiv.org/abs/1805.10820 (visited on 12/12/2018).

[46] Patrick Hall and Navdeep Gill. ***An Introduction to Machine Learning Interpretability-Dataiku Version***. O’Reilly Media, Incorporated, 2018.

[47] John H. Halton. “Algorithm 247: Radical-inverse Quasi-Random Point Sequence”. In: ***Communications of the ACM*** 7.12 (1964), pp. 701–702.

[48] Frederik Harder, Matthias Bauer, and Mijung Park. “Interpretable and Differentially Private Predictions”. In: ***Proceedings of the AAAI Conference on Artificial Intelligence***. Vol. 34. 04. 2020, pp. 4083–4090. ISBN: 2374-3468.

[49] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. ***The Elements of Statistical Learning: Data Mining, Inference, and Prediction***. Springer Science & Business Media, 2009. ISBN: 0-387-84858-4.

[50] Patrick J. Heagerty, Thomas Lumley, and Margaret S. Pepe. “Time-dependent ROC Curves for Censored Survival Data and a Diagnostic Marker”. In: ***Biometrics*** 56.2 (2000), pp. 337–344.

[51] Markus Herdin et al. “Correlation Matrix Distance, a Meaningful Measure for Evaluation of Non-Stationary MIMO Channels”. In: ***2005 IEEE 61st Vehicular Technology Conference***. Vol. 1. IEEE, 2005, pp. 136–140. ISBN: 0-7803-8887-9.

[52] Benjamin Hilprecht, Martin Härterich, and Daniel Bernau. “Monte Carlo and Reconstruction Membership Inference Attacks against Generative Models”. In: ***Proceedings on Privacy Enhancing Technologies*** 2019.4 (2019), pp. 232–249.

[53] Markus Hittmeir, Andreas Ekelhart, and Rudolf Mayer. “On the Utility of Synthetic Data: An Empirical Evaluation on Machine Learning Tasks”. In: ***Proceedings of the 14th International Conference on Availability, Reliability and Security***. 2019, pp. 1–6.

[54] Markus Hittmeir, Andreas Ekelhart, and Rudolf Mayer. “Utility and Privacy Assessments of Synthetic Data for Regression Tasks”. In: ***2019 IEEE International Conference on Big Data (Big Data)***. IEEE, 2019, pp. 5763–5772. ISBN: 1-72810-858-6.

[55] AI HLEG. “Ethics Guidelines for Trustworthy AI”. In: (2019). URL: https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai.

[56] Arthur E. Hoerl and Robert W. Kennard. "Ridge Regression: Biased Estimation for Nonorthogonal Problems". In: ***Technometrics*** 12.1 (Feb. 1970), pp. 55–67. ISSN: 0040-1706, 1537-2723. DOI: 10.1080/00401706.1970.10488634. URL: http://www.tandfonline.com/doi/abs/10.1080/00401706.1970.10488634 (visited on 08/02/2019).

[57] Chris Jay Hoofnagle, Bart van der Sloot, and Frederik Zuiderveen Borgesius. "The European Union General Data Protection Regulation: What It Is and What It Means". In: ***Information & Communications Technology Law*** 28.1 (2019), pp. 65–98.

[58] Sara Hooker et al. "A Benchmark for Interpretability Methods in Deep Neural Networks". In: ***Advances in neural information processing systems*** 32 (2019).

[59] Kurt Hornik. "Approximation Capabilities of Multilayer Feedforward Networks". In: ***Neural networks*** 4.2 (1991), pp. 251–257.

[60] Igor Ilic et al. "Explainable Boosted Linear Regression for Time Series Forecasting". In: ***Pattern Recognition*** 120 (2021), p. 108144.

[61] Gareth James et al. ***An Introduction to Statistical Learning***. Vol. 103. Springer Texts in Statistics. New York, NY: Springer New York, 2013. ISBN: 978-1-4614-7137-0 978-1-4614-7138-7. DOI: 10.1007/978-1-4614-7138-7. URL: http://link.springer.com/10.1007/978-1-4614-7138-7 (visited on 07/31/2019).

[62] John Johnston and John DiNardo. ***Econometric Methods***. Vol. 2. New York, 1972.

[63] James Jordon, Jinsung Yoon, and Mihaela van der Schaar. "Measuring the Quality of Synthetic Data for Use in Competitions". 2018. arXiv: 1806.11345.

[64] Ali Kassem et al. "Differential Inference Testing: A Practical Approach to Evaluate Sanitizations of Datasets". In: ***2019 IEEE Security and Privacy Workshops (SPW)***. IEEE, 2019, pp. 72–79. ISBN: 1-72813-508-7.

[65] Gajendra Jung Katuwal and Robert Chen. "Machine Learning Model Interpretability for Precision Medicine". Oct. 27, 2016. arXiv: 1610.09045 [q-bio]. URL: http://arxiv.org/abs/1610.09045 (visited on 05/12/2020).

[66] Been Kim, Rajiv Khanna, and Oluwasanmi O. Koyejo. "Examples Are Not Enough, Learn to Criticize! Criticism for Interpretability". In: ***Advances in neural information processing systems*** 29 (2016).

[67] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". 2014. arXiv: 1412.6980.

[68] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". 2013. arXiv: 1312.6114.

[69] Simon Kornblith et al. "Similarity of Neural Network Representations Revisited". In: ***International Conference on Machine Learning***. PMLR, 2019, pp. 3519–3529. ISBN: 2640-3498.

[70] Mark A. Kramer. "Nonlinear Principal Component Analysis Using Autoassociative Neural Networks". In: ***AIChE journal*** 37.2 (1991), pp. 233–243.

[71] Solomon Kullback and Richard A. Leibler. "On Information and Sufficiency". In: ***The annals of mathematical statistics*** 22.1 (1951), pp. 79–86.

[72] Miron B. Kursa, Aleksander Jankowski, and Witold R. Rudnicki. "Boruta–a System for Feature Selection". In: ***Fundamenta Informaticae*** 101.4 (2010), pp. 271–285.

[73] Thibault Laugel et al. "Defining Locality for Surrogates in Post-Hoc Interpretablity". 2018. arXiv: 1806.07498.

[74] Lenore J. Launer et al. "Body Mass Index, Weight Change, and Risk of Mobility Disability in Middle-Aged and Older Women: The Epidemiologic Follow-up Study of NHANES I". In: ***Jama*** 271.14 (1994), pp. 1093–1098.

[75] Jing Lei et al. "Distribution-Free Predictive Inference for Regression". In: ***Journal of the American Statistical Association*** 113.523 (2018), pp. 1094–1111.

[76] Benjamin Letham et al. "Constrained Bayesian Optimization with Noisy Experiments". In: ***Bayesian Analysis*** 14.2 (2019), pp. 495–519.

[77] Scott M. Lundberg, Gabriel G. Erion, and Su-In Lee. "Consistent Individualized Feature Attribution for Tree Ensembles". Feb. 11, 2018. arXiv: 1802.03888 [cs, stat]. URL: http://arxiv.org/abs/1802.03888 (visited on 03/10/2019).

[78] Scott M. Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: ***Advances in Neural Information Processing Systems***. Vol. 30. 2017.

[79] Scott M. Lundberg et al. "From Local Explanations to Global Understanding with Explainable AI for Trees". In: ***Nature machine intelligence*** 2.1 (2020), pp. 2522–5839.

[80] Edgar C. Merkle and Achim Zeileis. "Tests of Measurement Invariance without Subgroups: A Generalization of Classical Methods". In: ***Psychometrika*** 78.1 (2013), pp. 59–82.

[81] Tom Mitchell. ***Machine Learning***. McGraw Hill, 1997.

[82] Christoph Molnar. ***Interpretable Machine Learning***. Lulu. com, 2020. ISBN: 0-244-76852-8.

[83] Christoph Molnar. ***Limitations of Interpretable Machine Learning Methods***. 2020. URL: https://compstat-lmu.github.io/iml_methods_limitations/ (visited on 05/20/2020).

[84] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. "Methods for Interpreting and Understanding Deep Neural Networks". In: ***Digital signal processing*** 73 (2018), pp. 1–15.

[85] Guido F. Montufar et al. "On the Number of Linear Regions of Deep Neural Networks". In: ***Advances in neural information processing systems*** 27 (2014).

[86] Catarina Moreira et al. “An Investigation of Interpretability Techniques for Deep Learning in Predictive Process Analytics”. 2020. arXiv: 2002.09192.

[87] Andrew Ng and Michael Jordan. “On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes”. In: ***Advances in neural information processing systems*** 14 (2001).

[88] An-phi Nguyen and María Rodríguez Martínez. “On Quantitative Aspects of Model Interpretability”. 2020. arXiv: 2007.07584.

[89] Neha Patki, Roy Wedge, and Kalyan Veeramachaneni. “The Synthetic Data Vault”. In: ***2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)***. IEEE, 2016, pp. 399–410. ISBN: 1-5090-5206-2.

[90] Fabian Pedregosa et al. “Scikit-Learn: Machine Learning in Python”. In: ***the Journal of machine Learning research*** 12 (2011), pp. 2825–2830.

[91] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ““Why Should I Trust You?” Explaining the Predictions of Any Classifier”. In: ***Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining***. 2016, pp. 1135–1144.

[92] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “Anchors: High-precision Model-Agnostic Explanations”. In: ***Thirty-Second AAAI Conference on Artificial Intelligence***. 2018.

[93] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “Why Should i Trust You?: Explaining the Predictions of Any Classifier”. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2016, pp. 1135–1144. ISBN: 1-4503-4232-9.

[94] Mireia Ribera and Agata Lapedriza. “Can We Do Better Explanations? A Proposal of User-Centered Explainable AI.” In: ***IUI Workshops***. Vol. 2327. 2019, p. 38.

[95] Frank Rosenblatt. ***The Perceptron, a Perceiving and Recognizing Automaton Project Para***. Cornell Aeronautical Laboratory, 1957.

[96] Avi Rosenfeld and Ariella Richardson. “Explainability in Human–Agent Systems”. In: ***Autonomous Agents and Multi-Agent Systems*** 33.6 (2019), pp. 673–705.

[97] Donald B. Rubin. “Statistical Disclosure Limitation”. In: ***Journal of official Statistics*** 9.2 (1993), pp. 461–468.

[98] Steven L. Salzberg. ***C4. 5: Programs for Machine Learning by j. Ross Quinlan. Morgan Kaufmann Publishers, Inc., 1993***. Kluwer Academic Publishers, 1994. ISBN: 1573-0565.

[99] Wojciech Samek et al. “Evaluating the Visualization of What a Deep Neural Network Has Learned”. In: ***IEEE transactions on neural networks and learning systems*** 28.11 (2016), pp. 2660–2673.

[100] Gilbert Saporta. “Simultaneous Analysis of Qualitative and Quantitative Data”. In: ***Societa Italiana Di Statistica. XXXV Riunione Scientifica***. Vol. 1. CEDAM, 1990, pp. 62–72.

[101] Mattia Setzu et al. “Glocalx-from Local to Global Explanations of Black Box AI Models”. In: ***Artificial Intelligence*** 294 (2021), p. 103457.

[102] Sharath M. Shankaranarayana and Davor Runje. “ALIME: Autoencoder Based Approach for Local Interpretability”. In: ***International Conference on Intelligent Data Engineering and Automated Learning***. Springer, 2019, pp. 454–463.

[103] Lloyd S. Shapley. “A Value for N-Person Games”. In: ***Contributions to the Theory of Games*** 2.28 (1953), pp. 307–317.

[104] I. M. Sobol. “Points Which Uniformly Fill a Multidimensional Cube”. In: ***Mathematics, cybernetics series*** (1985), p. 32.

[105] Kacper Sokol and Peter Flach. “Explainability Fact Sheets: A Framework for Systematic Assessment of Explainable Approaches”. In: ***Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency***. 2020, pp. 56–67.

[106] Liwei Song, Reza Shokri, and Prateek Mittal. “Privacy Risks of Securing Machine Learning Models against Adversarial Examples”. In: ***Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security***. 2019, pp. 241–257.

[107] Charles Spearman. “The Proof and Measurement of Association between Two Things.” In: (1961).

[108] Carolin Strobl et al. “Conditional Variable Importance for Random Forests”. In: ***BMC bioinformatics*** 9.1 (2008), pp. 1–11.

[109] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. “Axiomatic Attribution for Deep Networks”. In: ***International Conference on Machine Learning***. PMLR, 2017, pp. 3319–3328. ISBN: 2640-3498.

[110] Dougal J. Sutherland et al. “Generative Models and Model Criticism via Optimized Maximum Mean Discrepancy”. 2016. arXiv: 1611.04488.

[111] Yi Tay et al. “Efficient Transformers: A Survey”. In: ***ACM Computing Surveys*** 55.6 (2022), pp. 1–28.

[112] Athanasios Tsanas et al. “Accurate Telemonitoring of Parkinson’s Disease Progression by Non-Invasive Speech Tests”. In: ***Nature Precedings*** (2009), pp. 1–1.

[113] Lao Tse. ***Credit Risk Dataset***. 2020. URL: https://www.kaggle.com/datasets/laotse/credit-risk-dataset (visited on 01/20/2023).

[114] Wessel N van Wieringen. “Lecture Notes on Ridge Regression”. In: ***arXiv preprint arXiv:1509.09169*** (2015). arXiv: 1509.09169.

[115] Wessel N. van Wieringen. “Lecture Notes on Ridge Regression”. July 22, 2019. arXiv: 1509.09169 [stat]. URL: http://arxiv.org/abs/1509.09169 (visited on 12/23/2019).

[116] Pierre-François Verhulst. “Correspondance Mathématique et Physique”. In: ***Ghent and Brussels*** 10 (1838), p. 113.

[117] Giorgio Visani et al. “Explanations of Machine Learning Predictions: A Mandatory Step for Its Application to Operational Processes”. In: (2019).

[118] Zhenchen Wang, Puja Myles, and Allan Tucker. “Generating and Evaluating Synthetic UK Primary Care Data: Preserving Data Utility & Patient Privacy”. In: ***2019 IEEE 32nd International Symposium on Computer-Based Medical Systems (CBMS)***. IEEE, 2019, pp. 126–131. ISBN: 1-72812-286-4.

[119] Chih-Kuan Yeh et al. “On the (in) Fidelity and Sensitivity of Explanations”. In: ***Advances in Neural Information Processing Systems*** 32 (2019).

[120] Muhammad Rehman Zafar and Naimul Mefraz Khan. “DLIME: A Deterministic Local Interpretable Model-Agnostic Explanations Approach for Computer-Aided Diagnosis Systems”. 2019. arXiv: 1906.10263.

[121] Eftim Zdravevski, Petre Lameski, and Andrea Kulakov. “Weight of Evidence as a Tool for Attribute Transformation in the Preprocessing Stage of Supervised Learning Algorithms”. In: ***The 2011 International Joint Conference on Neural Networks***. IEEE, 2011, pp. 181–188. ISBN: 1-4244-9637-3.

[122] Achim Zeileis, Torsten Hothorn, and Kurt Hornik. “Model-Based Recursive Partitioning”. In: ***Journal of Computational and Graphical Statistics*** 17.2 (2008), pp. 492–514.

[123] Alwin Yaoxian Zhang et al. “Development of a Radiology Decision Support System for the Classification of MRI Brain Scans”. In: ***2018 IEEE/ACM 5th International Conference on Big Data Computing Applications and Technologies (BDCAT)***. IEEE, 2018, pp. 107–115. ISBN: 1-5386-5502-0.

[124] Haozhe Zhang, Dan Nettleton, and Zhengyuan Zhu. “Regression-Enhanced Random Forests”. 2019. arXiv: 1904.10416.

[125] Xingyu Zhao et al. “Baylime: Bayesian Local Interpretable Model-Agnostic Explanations”. In: ***Uncertainty in Artificial Intelligence***. PMLR, 2021, pp. 887–896. ISBN: 2640-3498.

[126] Jianlong Zhou et al. “Evaluating the Quality of Machine Learning Explanations: A Survey on Methods and Metrics”. In: ***Electronics*** 10.5 (2021), p. 593.

[127] Yichen Zhou and Giles Hooker. “Interpreting Models via Single Tree Approximation”. Oct. 27, 2016. arXiv: 1610.09036 [stat]. URL: http://arxiv.org/abs/1610.09036 (visited on 12/12/2018).

[128] Hui Zou and Trevor Hastie. “Regularization and Variable Selection via the Elastic Net”. In: ***Journal of the royal statistical society: series B (statistical methodology)*** 67.2 (2005), pp. 301–320.