UNIVERSITAT DE BARCELONA

Facultat de Matemàtiques
i Informàtica

GRAU DE MATEMÀTIQUES

Treball final de grau

An introduction to explainable artificial intelligence with LIME and SHAP

Autor: Aleix Nieto Juscafresa

Directors: Dr. Albert Clapés
Dr. Sergio Escalera
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, June 2022

Abstract

Artificial intelligence (AI) and more specifically machine learning (ML) have shown their potential by approaching or even exceeding human levels of accuracy for a variety of real-world problems. However, the highest accuracy for large modern datasets is often achieved by complex models that even experts struggle to interpret, creating a tradeoff between accuracy and interpretability. These models are known for being "black box" and opaque, which is especially problematic in industries like healthcare. Therefore, understanding the reasons behind predictions is crucial in establishing trust, which is fundamental if one plans to take action based on a prediction, or when deciding whether or not to implement a new model. Here is where explainable artificial intelligence (XAI) comes in by helping humans to comprehend and trust the results and output created by a machine learning model. This project is organised in 3 chapters with the aim of introducing the reader to the field of explainable artificial intelligence. Machine learning and some related concepts are introduced in the first chapter. The second chapter focuses on the theory of the random forest model in detail. Finally, in the third chapter, the theory behind two contemporary and influential XAI methods, LIME and SHAP, is formalised. Additionally, a public diabetes tabular dataset is used to illustrate an application of these two methods in the medical sector. The project concludes with a discussion of its possible future works.

Resum

La intel·ligència artificial (IA), i més concretament, el *Machine Learning* (ML) han demostrat el seu potencial apropant-se, o fins i tot, superant els nivells de precisió humans per resoldre diversos problemes reals. Tanmateix, la precisió més alta per a grans conjunts moderns de dades sovint s'aconsegueix mitjançant models complexos que fins i tot costen d'interpretar als més experts, ja que creen un conflicte entre precisió i interpretabilitat. Aquests models són coneguts per ser "caixes negres" i opacs, cosa que és especialment problemàtica en indústries com la mèdica. Per tant, entendre els motius de les prediccions és crucial per generar confiança, cosa que és fonamental si es planeja actuar a partir d'una predicció i si es vol implementar o no un nou model. Aquí és on entra la *Explainable Artificial Intellgence* (XAI), que ajuda els humans a comprendre i a confiar en els resultats creats per un model de *Machine Learning*. Aquest projecte s'organitza en 3 capítols amb l'objectiu d'introduir el lector en el camp de la *Explainable Artificial Intellgence*. El *Machine Learning* i alguns conceptes que s'hi relacionen s'introdueixen al primer capítol. El segon capítol se centra en la teoria del model *Random Forest* que s'analitza en detall. Finalment, en el tercer capítol, es formalitza la teoria dels dos mètodes de XAI contemporanis i més influents, el LIME i el SHAP. A més, s'utilitza un conjunt de dades tabulars sobre la diabetis i públic per il·lustrar una possible aplicació d'aquests dos mètodes al sector mèdic. El projecte conclou amb una discussió sobre les seves possibles futures línies d'investigació.

2020 Mathematics Subject Classification. 68T01, 68T99

Acknowledgments

M’agradaria donar les gràcies a totes aquelles persones sense les quals ni aquest treball ni els meus estudis haguessin estat possibles.

A la meva família, concretament als meus pares, ja que sense ells no hauria arribat fins aquí.

Als amics que he fet al grau, especialment a en Guillem i en Pol. Gràcies per tots els moments compartits en el santuari.

Gràcies, Sergio, per acompanyar-me en aquest treball i per tots els consells.

Gràcies, Albert, per tot el temps que m’has dedicat i per introduir-me en el món de l’Explainable Artificial Intelligence. Ha sigut un plaer escoltar de tu.

Contents

Introduction

Machine learning (ML) is at the core of many recent scientific and technological advances. After computers started defeating experts at games such as Go (Silver, 2016)[1], many people have begun to wonder if computers might also make better drivers or even doctors.

In many machine learning applications, users are asked to trust a model to assist them in making decisions. However, a doctor would never operate on a patient because "the model said so". The most powerful machine learning models, based on deep neural networks, referred to as "black boxes", are hardly interpretable. Therefore, understanding the reasoning behind the model's predictions may assist users to decide whether or not to trust them.

When we deploy a machine learning model into production, we are essentially trusting it to make good predictions. Most of the time, to make that assessment, the model is evaluated by metrics in data we did not use to train the model. However, such metrics can be misleading, and the model may occasionally make embarrassing mistakes. Due to the fact that people often have strong intuition and business intelligence that is difficult to capture in evaluation metrics, understanding the model's predictions may be an additional useful tool for determining whether or not a model is reliable. This is precisely what motivated this project, understanding the reasoning behind the predictions of a model, and specifically, understanding why a prediction has been made, which does not necessarily mean being able to interpret the whole model. In order to achieve this, we formalise and unify the theory behind two XAI methods, LIME and SHAP, as well as the relationship between them. These methods have not been selected at random, but have been chosen because they both have had a big impact in the XAI field.

LIME (Local Interpretable Model-agnostic Explanations) is a revolutionary approach for interpreting and faithfully explaining individual predictions of any machine learning (Sarzynska, 2021)[2]. It reveals the reasoning behind a model's decision-making with the explanation. Then people will be able to see beyond the machine learning model and trust the predictions it makes. We will also explain SP-LIME, an algorithm that derives from LIME and which also helps to gain insight into how the model performs or to identify any pitfalls that may exist.

SHAP (SHapley Additive exPlanations) is also a novel way of explaining individual predictions (Lundberg, 2018)[3]. SHAP is based on the game theory optimal Shapley values. One of the attractions of this method is that it improves LIME in a particular way, a fact that will be studied in the project.

The second idea that has motivated the project is to see how these two methods are applied in the health industry, concretely in a scenario where we want to validate or reject a prediction. In these situations LIME and SHAP provide external tools to better understand the model predictions. To translate this idea into an experiment, a tabular database with patient health-related information has been used. The goal is to understand why a patient has been predicted as diabetic or non-diabetic. As we can see, we are in a problem of classification, which will be the ML task in which we will develop the project theory.

Therefore, the aim of this project is to introduce the reader to the field of explainable artificial intelligence with LIME and SHAP, both through theory and practical application. In order to progressively present the necessary concepts to comprehend these two methods completely, the project has been divided into the following 3 chapters:

1. **Machine learning preliminaries:** The machine learning concept and basic related terminology is introduced to the reader. Also, the problems that can be tackled using machine learning as well as the types of ML models are presented. Finally, the well-known bias-variance tradeoff is explained.

2. **Theory behind random forest:**. As already mentioned, the aim of the project is to understand the reasons for baseline predictions. However, we require a model to evaluate the explanations that will help us to achieve this. Random forest (RF) is the model selected and on which we will see an application of LIME and SHAP and it has been chosen for four reasons. The first is that random forest is an ensemble model, fact that makes it hardly interpretable, therefore, it will play the role of an arbitrary black box. The second is that it is a widely researched and highly performing method. This model captures complex distributions of the data by combining different hypothesis (submodels), concretely, combining various decision trees. This allows RF to create a robust model against overfitting. The third reason is because it inherently provides a way to compute feature importance, which will be of interest to us as we will be able to compare it to the explanations obtained through LIME and SHAP. Finally, the fourth reason is that it is a fast model to run because it requires low computational resources.

3. **Explainable AI:** This section begins with a presentation of the linear and weighted linear regression models. Next, the theory behind lasso and ridge regression regularization techniques is introduced, which will help us to reduce overfitting in a machine learning model. These concepts along with the previous two chapters are introduced to understand the concept of explainable artificial intelligence. First, XAI's taxonomy is presented and later the theory of LIME and the SHAP is formalized, where a theorem that relates them is presented. Finally we see how LIME and SHAP are applied through a public tabular diabetes database. This application exemplifies how the explanations of LIME and SHAP are observed and interpreted for a binary classification problem. In this case, the problem is to understand how the algorithm has classified a patient as diabetic or non-diabetic.

Eventually, the project concludes with a comparison of LIME and SHAP according two criteria. The first one refers to which situation is preferable to use one or the other and the second one to the points in which the both techniques differ. The project concludes with a comparison of LIME and SHAP and a discussion of future lines of research work.

1. Preliminaries

This chapter presents a preliminary introduction to machine learning, including an overview of some key learning tasks and applications, basic terminology, as well as an explanation of some fundamental concepts that we will come across as the project advances. The chapter is mainly based on (Mohri, 2018, Ch. 1)[4].

1.1 What is machine learning?

Machine Learning (ML) as a concept has been around for quite some time. Arthur Samuel, an American *IBMer* and pioneer in the field of computer gaming and artificial intelligence, was the first who coined the term machine learning in (Samuel, 1959)[5], when he designed a computer program for playing checkers.

Machine learning can be broadly defined as computational methods using *experience* to improve performance or to make accurate predictions. Here, experience refers to the past information available to the *learner*, which typically takes the form of electronic data collected and made available for analysis, and with this data the ML process looks for patterns in data so it can later make inferences based on the experience provided. This data could be in the form of digitised human-labelled training sets, or other types of information obtained via interaction with the environment. In all cases, its quality and size are crucial to the success of the predictions made by the learner.

An example of a learning problem is how to use a finite sample of randomly selected documents, each labelled with a topic, to accurately predict the topic of unseen documents. Clearly, the larger is the sample, the easier is the task. But the difficulty of the task also depends on the quality of the labels assigned to the documents in the sample, since the labels may not be all correct, and on the number of possible topics.

Machine learning consists of designing efficient and accurate prediction algorithms. As in other areas of computer science, some critical measures of the quality of these algorithms are their time and space complexity[1]. But, in machine learning, we will additionally need another notion of complexity, the *sample complexity*. Since the success of a learning algorithm depends on the data used, machine learning is inherently related to data analysis and statistics. Data-intensive machine-learning methods are increasingly being used in science, technology, and commerce, resulting in more evidence-based decision-making in a variety of fields such as health care, manufacturing, education, financial modelling, policing, and marketing.

As a summary, machine learning is a set of methods that computers use to make and improve predictions from experience based on data, without being explicitly programmed, combining fundamental concepts in computer science with ideas from statistics, probability and optimisation.

[1] Space complexity: Total amount of memory space required by an algorithm to solve an instance of the computational problem as a function of characteristics of the input.

1.2 Problems that can be tackled using machine learning

Machine learning has several applications, among which the following stand out:

- **Natural language processing (NLP):** This includes problems such as assigning a topic to a text or a document (text classification), determining automatically if the content of a web page is inappropriate or too explicit or spam detection. It also includes translation software, chatbots, grammar correction software, automatic summarising, voice assistants, and social media monitoring tools.
- **Speech processing applications:** This includes speech recognition, speech synthesis, speaker identification, language modelling and acoustic modelling.
- **Computer vision applications:** This includes object recognition, face detection. These applications are also important in the healthcare sector with techniques such as medical imaging or tumour classification.
- **Computational biology applications:** This includes protein function prediction as well as the analysis of gene and protein networks.

Many other problems such as fraud detection, learning to play games such as chess, unassisted control of vehicles, recommendation systems design or information extraction systems, are also tackled using machine learning techniques.

1.3 Some standard learning tasks

The following are some common ML *tasks*[2] that have been researched extensively:

- **Classification:** Problem of assigning a *category* to each item. For example, document classification consists of assigning a category such as politics, business, sports, or weather to each document, while image classification consists of assigning to each image a category such as car, train, or plane.
- **Regression:** Problem of predicting a real value for each item. Predicting stock prices or the price of a property based on property features are both examples of regression.
- **Ranking:** Problem of learning to order items according to some criterion. The canonical ranking example is returning web pages in a web search.
- **Clustering:** This is the problem of partitioning a set of items into homogeneous subsets. Clustering is often used to analyse very large datasets as it is an unsupervised learning method and it does not require *groundtruth*, therefore we avoid the task of labelling. For example, in the context of social network analysis, clustering algorithms attempt to identify natural communities within large groups of people.
- **Dimensionality reduction:** This problem consists of transforming an initial representation of items into a lower-dimensional representation while preserving as much information as possible. Principal component analysis is a well-known example.

[2] Task: Type of prediction or inference being made, based on the problem or question that is being asked, and the available data.

1.4 Terminology

From now on, a multitude of machine learning related concepts will show up. For this reason, providing a basic definition of some of the most relevant key concepts will assist in the comprehension of the following chapters.

- **Features:** Input variables used for prediction or classification.
- **Target variable:** It is the variable whose values, referred to as *labels*, are to be modelled and predicted by other variables. The target is usually called Y.
- **Prediction:** Target value "guessed" by the machine learning model based on the given features.
- **Instance:** It is a row in a dataframe. Other names for instance are: *data point*, *example*, *observation*. An instance consists of the feature values x_i and the target outcome y_i.
- **Dataset:** Collection of items that can be treated by a computer as a single unit for analytic and prediction purposes. In the case of tabular data, a dataset it called a *dataframe*, and corresponds to a spreadsheet that organises data into a 2-dimensional table of rows and columns. The rows are the instances and the columns are the features. The matrix with all the features is called X and each column is denoted by X_i. Across the project, dataframe will be referred as dataset and vice versa.
- **Training sample:** Instances used to train a learning algorithm.
- **Validation sample:** Instances used to tune the parameters of a learning algorithm when working with labelled data. The validation sample is used to select appropriate values for the learning algorithm's free parameters, called *hyperparameters*.
- **Test sample:** Instances used to evaluate the performance of a learning algorithm. The test sample is separate from the training and validation data and is not made available in the learning stage.
- **Loss function:** Using the previous notation, we denote Y the set of all labels, and Y' the set of possible predictions. A loss function $\mathcal{L}$ is a mapping $\mathcal{L} : Y \times Y' \to \mathbb{R}+$, that measures the difference, or loss, between a predicted and a true label. In most cases, $Y = Y'$ and the loss function is bounded, but these conditions do not always hold. Common examples of loss functions include the zero-one (or *misclassification*) loss defined over $\{-1, +1\} \times \{-1, +1\}$ by $\mathcal{L}(y, y') = \mathbb{1}_{\{y \neq y'\}}$ and the squared loss defined over $I \times I$ by $\mathcal{L}(y, y') = (y' - y)^2$, where $I \in \mathbb{R}$ is typically a bounded interval.
- **Hypothesis set:** A set of functions mapping features to the set of labels Y. Imagine we are in a spam email classification problem, these may be a set of functions mapping email features to $Y = \{\text{spam, non-spam}\}$. More generally, hypotheses may be functions mapping features to a different set Y', for example, they could be linear functions mapping email feature vectors to real numbers interpreted as scores ($Y' = \mathbb{R}$), with higher score values more indicative of spam than lower ones.

1.5 Types of machine learning methods

We will go over some popular machine learning scenarios in this section. The types of training data available to the learner, the order and method in which training data is received, and the test data used to evaluate the learning algorithm, all differ in these cases.

- **Supervised learning:** We can think of supervised learning in terms of function approximation, in which we train an algorithm and then choose the function that best describes the input data. Most of the time, we are unable to determine the true function that consistently makes good predictions among others, one reason is that the algorithm is based on human assumptions about how the computer should learn, which introduces bias. In Section 1.6 this term will be defined. Here, human specialists play the role of teacher, feeding the computer with training data (input/predictors) and showing it the correct answers (output/predictions), with the expectation that the machine will be able to learn patterns from the data. Supervised learning algorithms attempt to model relationships and dependencies between the target prediction output and the input features in order to predict the output values for new unseen data instances using the relationships learned from training data. The main types of supervised learning problems include regression and classification.
- **Unsupervised learning:** Unlabelled data is used to train the computer. There is no teacher here; in fact, after learning patterns in data, the computer may be able to teach us new things. This type of machine learning methods are commonly employed in pattern recognition and descriptive modelling, in fact, these algorithms are especially effective when the human expert is unsure what to search for in the data. There are no output categories or labels on which the algorithm can attempt to model relationships. These algorithms try to mine for rules, recognise patterns, summarise and aggregate data points in order to derive useful insights and better represent the data to consumers using techniques applied to the input data. The main types of unsupervised learning algorithms include clustering algorithms and association rule learning algorithms.
- **Semi-supervised learning:** In the preceding two types, either all of the observations in the dataframe have no labels or all of the observations have labels. Semi-supervised learning is somewhere in the middle. In a variety of contexts, the cost of labelling is relatively significant because it requires the use of qualified human experts. For example, to transcribe an audio segment or determine the 3D structure of a protein. In these cases, fully labelled training sets are infeasible, whereas acquisition of unlabelled data is relatively inexpensive. As a result, semi-supervised algorithms are the best options for model development in these situations. These methods take advantage of the fact that unlabelled data contains crucial information about group parameters, even if the group memberships are unknown.

- **Reinforcement learning:** The training and testing phases are intermixed in reinforcement learning. To collect information, the *agent*[3] actively interacts with the environment and in some cases affects the environment, and receives an immediate reward or punishment[4] for each action. The object of the agent is to maximise his reward over a course of actions and interactions with the environment. Some applications of the reinforcement learning algorithms are computer played board games (Chess, Go), robotic hands, and self-driving cars.

We could have used different criteria to classify types of machine learning algorithms. We stuck to the categorisation of (Fumo, 2017)[6], who uses the learning task, which is great to visualise the big picture of ML. Based on our data and the problem that we face, we can easily decide which of these methods to use to tackle our problem.

1.6 Bias-variance tradeoff

When performing statistical modelling, the goal is not to find a model that fits all of the training data points and has the minimum training data error. The objective is to give the model the ability to generalise well on new and unseen data.

When we discuss prediction models, prediction errors can be decomposed into two main subcomponents: error due to *bias* and error due to *variance*.

Bias: Error caused by erroneous assumptions in the learning algorithm, that skews the result of an algorithm in favour or against an idea. It is the difference between the average prediction of our model and the correct value which we are trying to predict. High bias can cause an algorithm to miss the relevant relations between features and target, oversimplifying the model due to paying little attention to the training data (*underfitting*).

Variance: Error caused by sensitivity to small fluctuations in the training set. A high variance model pays close attention to training data and does not generalise to data it has not seen before (*overfitting*).

Understanding these two types of error can help us diagnose model results and avoid the mistake of ovefitting or underfitting. The optimal situation is when our model has little bias and little variance, however, the problem comes when this does not happen and we want to improve one of these errors. The sweet spot where our machine model performs between the errors introduced by the bias and the variance is the *bias-variance tradeoff*.

[3] Agent: Artificial Intelligence (AI) algorithm that interacts with the environment.

[4] In Reinforcement Learning (RL), agents are trained on a reward and punishment mechanism. The agent is rewarded for correct moves and punished for the wrong ones. In doing so, the agent tries to minimise wrong moves and maximise the right ones.

A mathematical approach to the prediction error decomposition concept is given in (Fortmann, 2012)[7]:

If we denote the variable we are trying to predict as Y and our input space as X, we may assume that there is a relationship between them such as $Y = f(X) + \epsilon$ where the error term ϵ is normally distributed with a mean of zero like so $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$.

If we estimate a machine learning model $\hat{f}(X)$ of $f(X)$, the expected squared prediction error at a point x is:

$$Error(x) = \mathbb{E}[(Y - \hat{f}(x))^2].$$

This error may then be decomposed into bias and variance components:

$$\begin{aligned} Error(x) &= (\mathbb{E}[\hat{f}(x)] - f(x))^2 + \mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2 + \sigma_\epsilon^2 \\ &= Bias^2(\hat{f}(x)) + Var(\hat{f}(x)) + \sigma_\epsilon^2 \\ &= Bias^2 + Variance + Irreducible\ Error. \end{aligned}$$

That third term, irreducible error, is the noise term in the true relationship that cannot fundamentally be reduced by any model. Given the true model and infinite data to calibrate it, we would be able to reduce both the bias and variance terms to 0. However, in a world with imperfect models and finite data, there is a tradeoff between minimising the bias and minimising the variance.

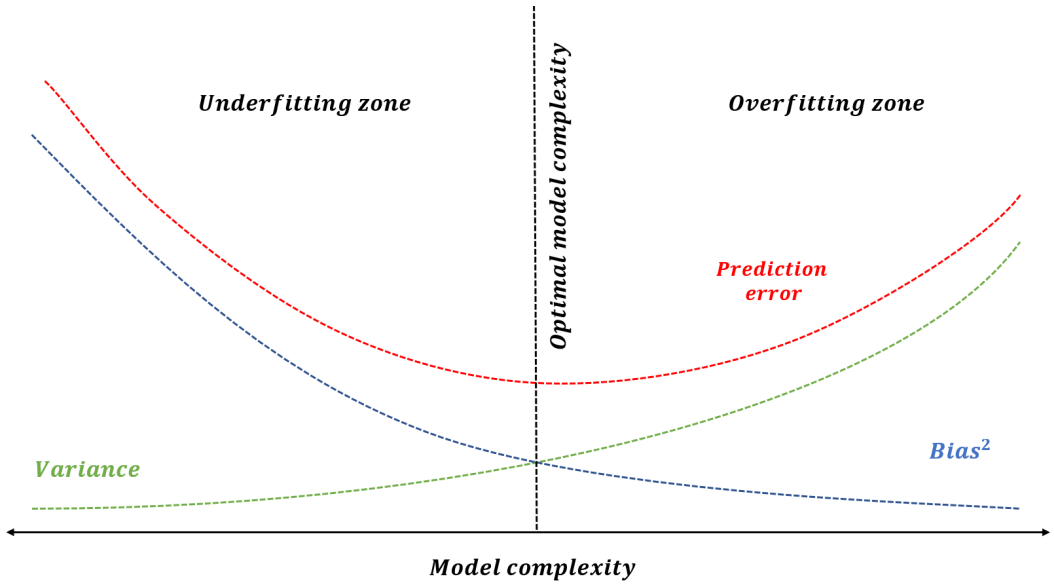


FIGURE 1: Bias and variance contributing to error.

As a model's number of parameters grows, the complexity of the model increases, fitting more noise in training data and leading to overfitting. This means that we are increasing the variance and decreasing the bias of the model. As we see in Figure 1, if the complexity falls short of the optimal point, we are in the underfitting zone, but if we keep raising the model's complexity, the prediction error will eventually the optimal spot and continue to increase, falling into the overfitting zone.

2. Theory behind random forest

The goal of this chapter is to introduce all the concepts which are required to construct a fairly complete explanation of the random forest model theory. Following the line of work, we will concentrate on the classification problem, starting with a theoretical introduction of a generic ML model. After that, the classification tree theory will be discussed, followed by the bootstrapping technique, and ending with the theory of ensemble methods, from which our target model, the random forest, will be derived.

2.1 Machine learning classification model

This section is based on (Mohri, 2018, Ch. 1.4)[4] and (Bourel, 2012)[8].

As seen in Section 1.4 we denote by X the set of all possible instances, also referred to as the input space, and by Y the set of all possible labels or target values. We assume that instances are *independent and identically distributed (i.i.d.)* according to some fixed but unknown distribution $F(X)$. To address the learning problem, we construct a function $f : X \to Y$ which belongs to a set of possible set of functions $\mathcal{H}$, called hypothesis set. We want f that given an input instance, is able to predict with accuracy the target variable $y = f(x)$. Given a labelled sample $\mathcal{S} = \{(x_1, y_1), ..., (x_n, y_n)\} = \{x, y\} \in (X \times Y)^n$ drawn i.i.d. according to $F(X)$, the task is to select a hypothesis $\hat{f} \in \mathcal{H}$ which minimises the risk $R_{\mathcal{L}}$ over a loss function $\mathcal{L}$ across our training sample $\mathcal{S}$. After this optimisation problem we obtain the function $\hat{f}$ among all the possible hypothesis in $\mathcal{H}$. This function $\hat{f}$, that maps inputs to predictions, will be our machine learning model.

In a classification problem, if $Y = 1, 2, \ldots, K$, the most used loss function is

$$\mathcal{L}(f, x, y) = \mathbb{1}_{\{y \neq f(x)\}}.$$

The *generalisation error* or risk of f is defined by

$$R_{\mathcal{L}}(f) = \mathop{\mathbb{P}}_{\{x,y\}\sim F(X)}(y \neq f(x)) = \mathop{\mathbb{E}}_{\{x,y\}\sim F(X)}(\mathcal{L}(f, x, y)) = \mathop{\mathbb{E}}_{\{x,y\}\sim F(x)}(\mathbb{1}_{\{y \neq f(x)\}}).$$

Thus, our optimal function $\hat{f}$, is the function $f \in \mathcal{H}$ at which the minimum of $R_{\mathcal{L}}(f)$ is attained. Mathematically:

$$\hat{f} = \mathop{\arg\min}_{f \in \mathcal{H}} R_{\mathcal{L}}(f) = \mathop{\arg\min}_{f \in \mathcal{H}} \mathop{\mathbb{E}}_{\{x,y\}\sim F(X)}(\mathbb{1}_{\{y \neq f(x)\}}).$$

The generalisation error of a hypothesis is not directly accessible to the learner since the distribution function $F(X)$ is unknown. However, the learner can measure the *empirical error* of a hypothesis on the labelled sample $\mathcal{S}$.

The empirical error or empirical risk of f is defined by

$$\hat{R}_{\mathcal{L},\mathcal{S}}(f) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{y_i \neq f(x_i)}.$$

Thus, the empirical error of $f \in \mathcal{H}$ is its average error over the sample $\mathcal{S}$, while the generalisation error is its expected error based on the distribution $F(X)$.

We now define the learning stages of our machine learning problem. Starting with a given collection of labelled examples, we first randomly partition the data into training, validation and test samples. Among other considerations, the amount of data reserved for validation depends on the number of hyperparameters of the algorithm.

Next, we associate relevant features to the examples, which is a critical step, since useful features can effectively guide the learning algorithm, whereas uninformative ones can be misleading. Although it is critical, to a large extent, the choice of the features reflects the user's prior knowledge about the learning task which in practice can have a dramatic effect on the performance results. Now, we use the features selected to train our learning algorithm by tuning the values of its hyperparameters. For each value of these parameters, the algorithm selects a different hypothesis out of the hypothesis set. We choose the one resulting in the best performance on the validation sample.

Finally, using that hypothesis, we predict the labels of the examples in the test sample. The performance of the algorithm is evaluated by a *metric* that compares the predicted and true labels. In classification, a commonly used metric is *accuracy*, which is defined as the fraction of predictions our model got right. For regression, other metrics such as *MSE* or *MAE* are used. Test metrics do not have to match the loss used to measure train error and validation during the learning stage. Thus, the performance of an algorithm is, of course, evaluated based on its test error and not on its error in the training sample.

2.2 Decision tree

The information of this section has been extracted from (Hastie, 2009, Ch. 9.2)[9] complemented with (Lee, 2020)[10] and (López, 2019)[11].

Many machine learning models fail in situations where the relationship between features and outcome is nonlinear or where features interact with each other. It is time for the *Decision Tree (DT)* to shine. Tree based models progressively split the data multiple times based on certain feature cutoff values until they reach sets that are small enough to be described by some label, with each instance belonging to one of them. In each split, the decision tree tries to split the data into two or more groups (*nodes*), so that the groups are as heterogeneous as possible from each other, and the data points that fall into the same group are homogeneous. For a multitude of reasons, decision trees are immensely popular, the most notable of which being their interpretability. They can be trained quickly and easily, which expands their potential well beyond scientific context. Regarding the DT structure, *terminal or leaf nodes* are the ultimate subsets, whereas *internal nodes or split nodes* are the intermediate subsets *(See Appendix A)*.

Trees can be used for classification and regression and there are various algorithms that can grow a tree. They differ in the possible structure of the tree (e.g. number of splits per node), the criteria how to find the splits, when to stop splitting and how to estimate the simple models within the leaf nodes. The classification and regression trees (*CART*) algorithm is probably the most popular algorithm for tree induction. We will focus on CART, but the interpretation is similar for most other tree types.

2.2.1 CART algorithm

CART is a decision tree algorithm that produces binary classification or regression trees, depending on whether the target variable is categorical or numeric, respectively. It can use the same variables more than once in different parts of the same DT, which may uncover complex interdependencies between sets of variables. Our data consists of p inputs and a response variable taking values $1, 2, \ldots, K$ for each of n observations: that is, (x_i, y_i) for $i = 1, 2, \ldots, n$, with $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$.

Our decision tree needs to automatically decide on the splitting variables and split points, in other words, we need to determine the different subsets across the tree. We will do this introducing a measure of node *impurity*.

First, we define the proportion of class k observations in node m as follows:

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} \mathbb{1}_{\{y_i = k\}}.$$

where R_m represents the subset of instances in node m, and N_m the number of observations in this node.

One of the most popular metrics for measuring node impurity is called *Gini impurity index*, but there are also well-known sounding methods like *entropy* and *information gain*. However, numerically, the methods are all quite similar *(See Appendix B)*. The Gini index for a node m is defined as follows:

$$I(m) = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) = 1 - \sum_{k=1}^{K} \hat{p}_{mk}^2, \qquad 0 \leq I(m) \leq 0.5.$$

Gini index gives an idea of how "impure" the node is. When all the instances in the node have the same label, there is a perfect classification and our node is maximally pure ($I(m) = 0$). Contrarily, when all the instances are equally distributed among different labels, we face the worst case and the node is totally impure ($I(m) = 0.5$).

Suppose we have reached the leaf node m, which contains N instances. For each feature, we calculate the Gini index for the two split nodes m_1 and m_2 that contain N_1 and N_2 observations respectively. The Gini impurity index for one feature A in the node m will be the weighted average of the leaf node impurities:

$$I_A(m) = \frac{N_1}{N} I(m_1) + \frac{N_2}{N} I(m_2).$$

We first start at the very top of our tree, at the *root node*. And then the algorithm selects the feature for splitting that would result in the best partition in terms of the Gini index and adds this split to the tree. Note that if the node itself has the lowest Gini index score, then there is no point in continuing separating the instances according to any other variable, and this node becomes a leaf node. The algorithm continues this search-and-split recursively in both new nodes until a stop criterion is reached.

Stopping criterion

Possible stopping criteria are the minimum number of instances that have to be in a node before the split, or the minimum number of instances that have to be in a terminal node. If the count is less than some minimum then the split is not accepted and the node is taken as a final leaf node.

The stopping criterion is important as a very large tree might overfit the data, while a small tree might underfit, not capturing the underlying distribution of the training data.

Pruning

The complexity of a decision tree is defined as the number of splits in the tree. Simpler trees are preferred, as they are easy to interpret and less likely to overfit the data. To avoid overfitting, once our decision tree has been trained, one can use *pruning* to lift its performance.

The fastest and simplest pruning method is to work through each leaf node in the tree and evaluate the effect of removing it using the validation set. Leaf nodes are removed only if it results in a drop in the overall cost function on this set. We stop removing nodes when no further improvements can be made.

A more sophisticated approach to pruning a tree is using *cost-complexity pruning*, which we now describe. We define a subtree $T \subset T_0$ to be any tree that can be obtained by pruning our tree T_0, that is, collapsing any number of its internal (non-terminal) nodes. Let $|T|$ denote the number of terminal nodes in T. We define the *cost complexity criterion*:

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m I(m) + \alpha|T|.$$

The tuning parameter $\alpha \geq 0$ governs the tradeoff between tree size and its goodness of fit to the data, and it is estimated via cross-validation. The idea is to find, for a given α, the subtree $T_\alpha \subseteq T_0$ to minimise $C_\alpha(T)$. Large values of α result in smaller trees T_α, and conversely for smaller values of α. As the notation suggests, with $\alpha = 0$ the solution is the full tree T_0. For each α one can show that there is a unique smallest subtree T_α that minimises $C_\alpha(T)$[1].

[1] See (Breiman, 1984)[12] or (Ripley, 1996)[13].

Non binary features

So far we have seen how to build a tree with "yes/no" questions at each step, but what if we have numerical data or categorical variables with more than two labels?

- **Numerical features**

  For numerical data we first sort the feature values, from lowest to highest. Then we calculate the average value for all pairs of subsequent instances and finally we select the lowest Gini impurity resulting from splitting the feature in these average points.

- **Multi-label categorical features**

  For categorical features having q possible unordered labels, there are $2^{q-1} - 1$ possible partitions[2] of the q values into two groups. We select the split with the lowest Gini impurity, however, for large q the computation of Gini Index for each possible split becomes prohibitive. However, CART handles it without exponential complexity, but the algorithm it uses to do so is highly non-trivial.[1]

Why binary splits?

Rather than splitting each node into just two groups at each stage as we have done, we might consider multiway splits into more than two groups. While this can sometimes be useful, it is not a good general strategy. The problem is that multiway splits fragment the data too quickly, leaving insufficient data at the next level down. Hence we would want to use such splits only when needed. Since multiway splits can be achieved by a series of binary splits, the latter are preferred.

Limitations

Trees fail to deal with linear relationships. Any linear relationship between an input feature and the outcome has to be approximated by splits, creating a step function.

Decision trees tend to overfit on their training data, which means that they suffer from high variance. Few changes in the training dataset can create a completely different tree.

They suffer from inherent instability, since due to their hierarchical nature, the effect of wrongly selecting a feature in the top splits propagates down this error to all of the splits below. It does not create confidence in the model if the structure changes so easily.

DTs can also create biased trees if some classes dominate over others. This is a problem in *unbalanced datasets* (where different classes in the dataset have different number of observations). That is why it is important to balance the dataset before building the DT.

Decision tree splitting algorithms cannot see much further than the present point of development, they are *greedy*, which means that they look for a locally optimal and not

[2] For each categorical value, the number of possible subsets of a set of size q is 2^q, because of the symmetry between left and right we have the half of subsets, $\frac{2^q}{2} = 2^{q-1}$. Because the empty set to either side of the split is not allowed we get $2^{q-1} - 1$.

a globally optimal at each step. Trees grow by adding one node at a time, and do not implement any backtracking technique.

2.3 Bootstrapping

This section has been based on (Hastie, 2009, Ch. 7.11)[9] and (Rocca, 2019)[14] complemented with (Frost, 2018)[15] and (Llano, 2020)[16].

From our training sample $\mathcal{S}$, we randomly draw datasets with replacement from the training data, each new dataset $\mathcal{S}^*$ the same size as the original training set. This method is known as *bootstrapping*.

As the true population distribution is unknown, the error in a sample statistic against its population value is unknown. The basic idea of bootstrapping is that inferencing the properties of a population from sample data can be done by resampling the sample data and performing inference about this resampled data. Here we are treating our sample as the population, as a result, the essential premise of bootstrapping is that the original sample properly reflects the whole population.

Traditional hypothesis testing procedures require equations that estimate sampling distributions using the properties of the sample data, the experimental design, and a test statistic. To obtain valid results, one will need to use the proper test statistic and satisfy the assumptions. The bootstrap method uses a very different approach to estimate sampling distributions. Since the bootstrapping procedure is distribution-independent it provides an indirect method to assess the properties of the distribution underlying the sample and the parameters of interest that are derived from this distribution.

Let us study the number of instances of the original training dataset that a bootstrap dataset $\mathcal{B}$ will have. To do it we define the following probability space $(\Omega, \mathcal{F}, \mathbb{P})$:

- Ω : The sample space is our training sample $\mathcal{S}$.
- $\mathcal{F}$**:** As Ω is a finite set of n independent observations we have that the $\sigma - algebra$ $\mathcal{F}$ is formed of all the possible subsets of Ω, that is $\mathcal{F} = \mathcal{P}(\Omega)$.
- $\mathbb{P}$: Our events of interest $\mathcal{E}$ occur when we select any instance $z_i = (x_i, y_i)$ to create our bootstrap dataset B. Therefore, the probability measure $\mathbb{P}\colon \mathcal{F} \longrightarrow [0,1]$ can be defined as $\mathbb{P}(\mathcal{E}) = \mathbb{P}(z_i \in B)$.

Once our probability space has been defined, we may imagine that the construction of our bootstrap sample $\mathcal{B}$ is equivalent to an experiment of n trials. In every trial we select one instance uniformly at random with replacement, therefore the probability that we select an instance z_i is $\mathbb{P}(\mathcal{E}) = \frac{|\mathcal{E}|}{|\Omega|} = \frac{1}{n}$, and the probability we do not select this instance is $\mathbb{P}(\mathcal{E}^c) = 1 - \frac{1}{n}$. Then the probability that an instance z_i is not selected in the n selections for our bootstrap sample is:

$$\left(\bigcup_{i=1}^{n} \mathbb{P}(\mathcal{E})\right)^c = \bigcap_{i=1}^{n} \mathbb{P}(\mathcal{E}^c) = \left(1 - \frac{1}{n}\right)^n.$$

Therefore, the probability that z_i is included at least once in the bootstrap dataset $\mathcal{B}$ is $1-(1-\frac{1}{n})^n$. Taking the limit as n approaches infinity and $e = \lim_{n\to\infty}(1+\frac{1}{n})^n$ we get:

$$\lim_{n\to\infty} 1-\left(1-\frac{1}{n}\right)^n = 1-\lim_{n\to\infty}\left(1-\frac{1}{n}\right)^n = 1-\lim_{n\to\infty}\left[\left(1+\frac{1}{-n}\right)^{-n}\right]^{\frac{1}{-n}\cdot n} = 1-\frac{1}{e}.$$

For a big n, the probability of an element $z_i \in \mathcal{S}$ being selected for the bootstrap dataset is $1-\frac{1}{e} \approx 0.63$. Therefore, each bootstrap dataset contains $n' \approx 0.63\,n$ instances of the original training dataset. The remaining instances form the *Out-Of-Bag Dataset* and will be used later to measure the model accuracy *(Out-Of-Bag Error)*.

2.4 Ensemble methods

This section is based on (Bourel, 2012)[8], (Hastie, 2009, Ch. 15)[9] and (Rocca, 2019)[14].

There is a variety of techniques for addressing the previously discussed decision tree limitations. The *ensemble methods* consist of building a number of predictors, which we call *intermediate hypotheses*, from a single set of data and combining them in some way to obtain a more stable predictor with improved performance while reducing variance.

Suppose we are facing a multi-class classification problem where our response variable takes K possible values (labels). If we have our training dataset $\mathcal{S}$, in order to obtain more robust prediction models, we can build M classifiers $g_1, g_2, \ldots, g_M$ to predict the dependent variable Y and combine them. An *aggregate classifier* can be defined naturally, as the majority vote of the M intermediate classifiers, that is:

$$f(x) = \underset{k\in\{1,\ldots,K\}}{\arg\max}\left(\sum_{m=1}^{M} \mathbb{1}_{\{g_m(x)=k\}}\right).$$

We can also consider the probabilities of each class returned by all the models, average these probabilities and keep the class with the highest average probability (*soft-voting*).

Another way to combine these intermediate classifiers can be by a weighted vote. If $\alpha_1, \ldots, \alpha_M$ are real numbers, we can define the aggregate classifier as the one that predicts the class resulting from the largest weighted sum of the classifiers $g_1, g_2, \ldots, g_M$:

$$f(x) = \underset{k\in\{1,\ldots,K\}}{\arg\max}\left(\sum_{m=1}^{M} \alpha_m \mathbb{1}_{\{g_m(x)=k\}}\right).$$

That is, we construct an aggregate model that predicts the class that maximises the weighted sum of the classifiers $g_1, g_2, \ldots, g_M$.

Ensemble methods can be divided into two large families: *homogeneous model ensemble methods* and *heterogeneous model ensemble methods*. The methods that combine hypotheses from the same type of learning algorithm belong to the first family. An example could be combining a number of classification trees or neural networks. *Bagging*, *random forest*, and *boosting* are the most popular in this category.

The family of heterogeneous ensemble techniques is composed of ensemble methods that incorporate hypotheses from several learning algorithms, such as merging three classification trees, a support vector machine, and a neural network. A good example is the *stacking* method.

2.4.1 Bagging

Bagging, or **B**ootstrap **Agg**regating, gets its name because it combines bootstrapping and aggregation to form one ensemble model. It was first introduced in (Breiman, 1996)[17] and it is a method of aggregating homogeneous models based on the majority vote or the average in the regression case.

Due to the theoretical variance of the training dataset (we remind that our dataset is an observed sample coming from a true unknown underlying distribution), the fitted model is also subject to variability: if another dataset had been observed, we would have obtained a different model. That is the variance of the model.

The idea of bagging is then simple: we want to fit several independent models and "average" their predictions in order to obtain a model with a lower variance. However, we cannot, in practice, fit fully independent models as it would require too much data. So, we rely on the good "approximate properties" of bootstrap samples to fit models that are almost independent.

Roughly speaking, as the bootstrap samples are approximately independent and identically distributed (i.i.d.), so are the learned base models. Intuitively we can understand it in this way. Suppose we have n random variables $X_1, X_2, \ldots, X_n$ which are independent and identically distributed drawn from a distribution of expected value given by μ and finite variance given by σ^2, then the *Central Limit Theorem* states that:

$$\bar{X}_n \xrightarrow{\mathcal{L}} \mathcal{N}(\mu, \frac{\sigma^2}{n}). \tag{2.1}$$

In other words, what this means is that for n large enough, $\bar{X}_n$ converges in law to a normal distribution $\mathcal{N}(\mu, \frac{\sigma^2}{n})$.

Here we can see that there is reduction in variance when there is an aggregation happening. Then, "averaging" weak learners outputs do not change the expected answer but reduce its variance, leading to an improvement of the accuracy.

Suppose we fit a model to our training data $\mathcal{S} = \{(x_1, y_1), ..., (x_n, y_n)\}$, obtaining the prediction $\hat{f}(x)$ at input x. Bagging technique is built in three stages:

- (1) From our training data, we build M bootstrap datasets $\mathcal{S}^*_m$, $m = 1, \ldots, M$.
- (2) For each bootstrap sample $\mathcal{S}^*_m$, $m = 1, \ldots, M$, we fit our model giving prediction $\hat{f}^*_m(x)$.
- (3) The bagging estimate is defined by

$$\hat{f}_{bag}(x) = \underset{k \in \{1,\ldots,K\}}{\arg\max} \left(\sum_{m=1}^{M} \mathbb{1}_{\{\hat{f}^*_m(x)=k\}} \right).$$

Bagging allows us to construct aggregate classifiers using any model we want; however, the scenario that interests us is when each model is a decision tree, which we will refer to as *bagged trees*.

For classification we can understand bagging as a consensus of independent *weak learners* (Diettrich, 2000)[18]. Let $g(x) = 1$ the correct prediction in a two-class classification example. Suppose each of the weak learners g_i , $i = 1, 2, \ldots, M$ has a success rate $p > 0.5$ and let $S(x) = \sum_{m=1}^{M} \mathbb{1}_{\{g_m=1\}}$ be the consensus vote for class 1. Since the weak learners are assumed to be independent, $S(x) \sim Bin(M, p)$. Therefore, we know that

$$\lim_{M\to\infty} \mathbb{P}(S(x) \leq M/2) = \lim_{M\to\infty} \sum_{i=1}^{\lfloor \frac{M}{2} \rfloor} \binom{M}{i} p^i (1-p)^{M-i} = 0,$$

hence $\mathbb{P}(S(x) > M/2) = 1 - \mathbb{P}(S(x) \leq M/2) \to 1$ when $M \to \infty$ *(See Appendix C)*.

Here we have shown that if the probability of each aggregate classifier is bigger than 0.5, then the probability that the correct class is the most voted tends to 1 as the number of classifiers tends to infinity. This concept has been popularised outside of statistics as the "*Wisdom of Crowds*" - "the collective knowledge of a diverse and independent body of people typically exceeds the knowledge of any single individual, and can be harnessed by voting", (Surowiecki, 2004)[19]. Of course, the main caveat here is the independence between the aggregate models because bagged trees are not independent. For example, imagine that there is one very strong predictor within the data. In each tree, this strong predictor will likely be the first split. Therefore, the prediction of most trees will be similar. In other words, the predictions will be correlated and hence not independent.

Finally, we can mention that one of the big advantages of bagging is that it can be parallelised. As the different models are fitted independently from each other, intensive parallelisation techniques can be used if required.

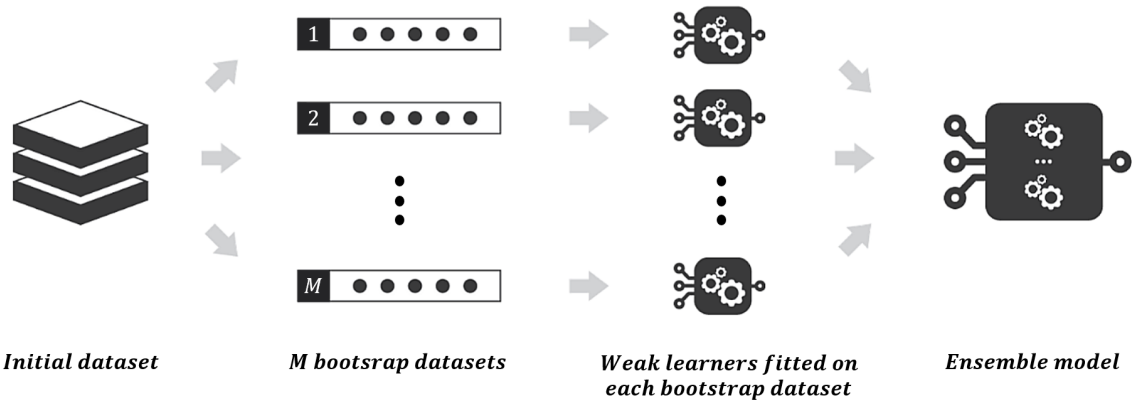


FIGURE 2: Bagging entails fitting numerous base models to various bootstrap samples and constructing an aggregate model that "averages" the outputs of these weak learners, adapted from (Rocca, 2019)[14].

2.4.2 Random forest

Random forest (RF) algorithm was proposed in (Breiman, 2001)[20], and it combines the techniques of CART and bagging with a substantial modification.

Trees are ideal candidates for bagging, since they can capture complex interaction structures in the data, and if grown sufficiently deep, have relatively low bias. Since trees are notoriously noisy, they benefit greatly from the majority vote. Moreover, since each tree generated in bagging is identically distributed (i.d.), the expectation of an average of such M trees is the same as the expectation of any one of them. This means the bias of bagged trees is the same as that of the individual trees, and the only hope of improvement is through variance reduction. As seen in the *Central Limit Theorem*, an average of M i.i.d. random variables, each with variance σ^2 , has variance σ^2/M. In the case of bagged trees, as they are simply i.d., trees may not be independent when there is a variable with strong predictive power. Suppose each tree has variance σ^2 and the correlation between two trees is $\rho > 0$. Then, in the regression scenario, the variance of the average of M trees is:

$$
\begin{aligned}
&Var\left(\frac{1}{M}\sum_{m=1}^{M}\hat{f}_m^*(x)\right) = \frac{1}{M^2}\sum_{m=1}^{M}\sum_{l=1}^{M}Cov(\hat{f}_m^*(x),\hat{f}_l^*(x)) \\
&= \frac{1}{M^2}\sum_{m=1}^{M}\sum_{l\neq m}^{M}\left(Cov(\hat{f}_m^*(x),\hat{f}_l^*(x)) + Var(\hat{f}_m^*(x))\right) \\
&= \frac{1}{M^2}\sum_{m=1}^{M}((M-1)\rho\sigma^2+\sigma^2) = \frac{M((M-1)\rho\sigma^2+\sigma^2)}{M^2} \\
&= \frac{M^2\rho\sigma^2 + M\sigma^2(1-\rho)}{M^2} = \rho\sigma^2 + \frac{\sigma^2(1-\rho)}{M}.
\end{aligned}
$$

As $M \to \infty$, the second term disappears, and the variance of the model depends uniquely on $\rho\sigma^2$. The idea in random forest is to improve the variance reduction of bagging by reducing the correlation between the trees, without increasing the variance too much, which will lead to a reduction in the overall model variance.

Here is where the modification of bagging comes into play, as we create a huge collection of *de-correlated* trees. When using bootstrap samples, at the splitting node step, we have the full disposal of features to choose from, which will cause that the data will mostly split off at the same features throughout each model. In order to correct this and reduce the correlation between trees, in random forest, instead of only sampling over the observations in the dataset to generate a bootstrap sample, we also sample over features and keep only a random subset of them to build the tree.

Emphasise that the trees obtained are maximal, that is, they are not pruned, having low bias but high variance and so are appropriate candidates for the bagging method that is mainly focused at reducing variance.

Sampling over features has indeed the effect that all trees do not look at the exact same information to make their decisions, leading to more diversification and reducing the correlation between the different returned outputs. Another advantage of sampling over the features is that it makes the decision-making process more robust to missing data: observations (from the training dataset or not) with missing data can still be regressed or classified based on the trees that take into account only features where data are not missing. Thus, the random forest algorithm combines the concepts of bagging and random feature subspace selection to create more robust models.

The low correlation between trees is the key, as a large number of relatively uncorrelated trees operating as a committee will outperform any of the individual constituent models. The reason for this wonderful effect is that the trees protect each other from their individual errors. While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction.

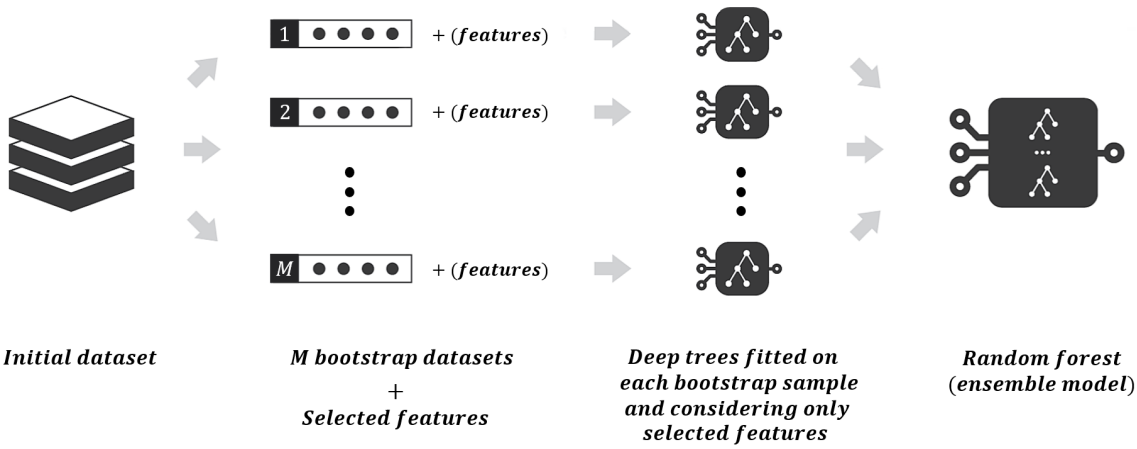


FIGURE 3: Random forest model uses trees as weak learners in a bagging process. Each tree is built using bootstrapping considering only a subset of variables chosen at random, adapted from (Rocca, 2019)[14].

Random forest has been shown to be one of the algorithms with better performances in learning problems, in particular in those that have a significant number of explanatory variables. However if only a few are of relevance, boosting is usually more effective than random forest.

The out-of-bag error estimate

As we saw in bootstrapping, replacement causes that we do not select all the instances in our bootstrap datasets. The instances left out in the creation of our bootstrap datasets can be aggregated, forming the out-of-bag dataset.

The Out Of Bag (OOB) error is a way of validating the random forest model, i.e, measuring its prediction error. OOB error is the mean prediction error on each training sample x_i of the out-of-bag dataset, using only the trees that did not contain x_i in their bootstrap sample.

Since each out-of-bag set is not used to train the model, it is a good test for the performance of the model and the OOB is calculated as follows:

1. Find all trees that are not trained by the OOB instance.

2. We run this out-of-bag instance through all the trees that were built without it. We assign to this instance the label with the most votes through all these trees.

3. Repeating steps 1 and 2 for all out-of-bag instances, we can measure how accurate our random forest is by the proportion of out-of-bag samples that were correctly classified by the random forest, which is the out-of-bag error.

In RF, there is no need for cross-validation or a separate test set to get an unbiased estimate of the test set error. Over many iterations, OOB error and cross-validation should produce a very similar error estimate, so starting with a high number of iterations is a good idea. That is, once the OOB error stabilises, it will converge to the cross-validation (specifically leave-one-out cross-validation) error. The advantage of the OOB method is that it requires less computation and allows us to test the model as it is being trained.

Number of features randomly selected

As we have seen, at each node split, we select $q \leq p$ of the input variables at random as candidates for splitting. RF inventors suggest as a default value for classification, $q = \lfloor \sqrt{p} \rfloor$. However, now that we know how to estimate the accuracy of the random forest, we can talk about how to select the best value for this parameter, treating it as a hyperparameter. We test a bunch of different values and choose the most accurate RF.

Feature importance

Finally, we explain two criteria for determining the importance of each of the explanatory features in the random forest model:

- **Gini importance**

  Every time a split of a node m is made on a feature A, the Gini index for this feature is less than the Gini index for the current node. This difference is called *Gini gain* and is defined as follows:

  $$\Delta I(A) = I(m) - I_A(m).$$

  By adding up the Gini gains for each individual feature over all trees in the RF, we obtain a fast feature importance that is often very consistent with the permutation importance measure.

- **Permutation importance**

  The increase in the model's prediction error after permuting the feature's values, is measured by permutation feature importance. The concept is quite simple: we calculate the increase in the model's prediction error after permuting a feature to determine its relevance. Because the model relies on the feature for prediction, a feature is "important" if shuffling its values increases the model error. A feature is

"unimportant" if shuffling its values has no effect on the model error, because the feature was ignored for prediction in this situation. According to (Molnar, 2022, Ch. 8.5)[21] test data should be used to compute the feature importance.

When the b^{th} tree is grown, the OOB samples are passed down the tree, and the prediction accuracy is recorded. Then, the values for the j^{th} variable are randomly permuted in the OOB samples, and the accuracy is again computed. The accuracy decrease caused by this permuting is averaged across all trees, and it is used as a measure of variable j's importance, given as a percentage of the maximum.

When variables are highly correlated, it has been shown in (Parr, 2018)[22] that permutation importance provides more robust estimates than Gini importance. Thus, instead of interpreting internal model parameters as proxies for feature significance, this work recommends using permutation importance for all machine learning models.

It is important to note that this method not only works for random forests as it does not depend on the model in question as only the change in prediction is valued. Therefore this method is said to be *model-agnostic*, i.e., it can be applied to any machine learning model.

3. Explainable artificial intelligence

In this chapter we formalise and unify the theory behind two modern and relevant XAI methods, LIME and SHAP. We also compare the global relevance of the random forest features presented in the previous chapter to local explanations given by these models[1]. All of this is performed by using a diabetes database, from which numerous conclusions are drawn.

To be able to fully understand LIME and SHAP, we first introduce some necessary background by presenting and analysing linear regression and weighted linear regression algorithms. We also present L1 and L2 regularisation methods that will prevent high complexity ML models from overfitting.

3.1 Linear regression

Simple linear regression is a statistical approach for describing relationships between two continuous (quantitative) variables. This section covers the basics of simple linear regression and it has been based on (Hastie, 2009, Ch. 3)[9] and (Molnar, 2022, Ch. 5)[21] complemented with (Simon and Young, 2022)[23].

Simple linear regression gets its adjective "simple" because it concerns the study of only one predictor variable. In contrast, multiple linear regression, which we will not cover in this project, gets its adjective "multiple" because it concerns the study of two or more predictor variables.

Linear models can be used to model the dependence of a regression target on some features X. Following the notation introduced in Section 1.4, we have a vector of features $X = (X_1, X_2, ..., X_p)$, and want to predict a real-valued output Y. The linear regression model has the form

$$Y = f(X) + \epsilon = \beta_0 + \sum_{j=1}^{p} \beta_j X_j + \epsilon. \tag{3.1}$$

The predicted outcome of an instance is a weighted sum of its p features. The betas (β_j) represent the learned feature *weights* or *coefficients*. The first weight in the sum (β_0) is called the *intercept* and it is not multiplied with a feature. The intercept is the expected mean value of Y when the vector of features is 0. The interpretation of the intercept is usually not relevant because instances with all features values at zero often make no sense. The interpretation is only meaningful when the features have been standardised (mean of zero, standard deviation of one). Then the intercept reflects the predicted outcome of an instance where all features are at their mean value. The epsilon (ϵ) is the error we still make, i.e. the difference between the prediction and the actual outcome. Later we will see some model assumptions regarding this error.

[1] Global refers to a full explanation of the model while local refers to the explanation of the prediction for an individual instance.

To train the model, we receive a training dataset $\mathcal{S} = \{(x_1, y_1), ..., (x_n, y_n)\}$ from which to estimate the weights β. Each $x_i = (x_{i1}, \ldots, x_{in})$ is a vector of feature measurements for the i^{th} instance. The most popular estimation method is *least squares*, in which we pick the coefficients $\beta = (\beta_0, \ldots, \beta_p) \in \mathbb{R}^n$ to minimise the residual sum of squares (RSS):

$$\hat{\beta}_{OLS} = \arg\min \text{RSS}(\beta) = \underset{\beta_0,...,\beta_p}{\arg\min} \sum_{i=1}^{n}(y_i - (\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}))^2 = \underset{\beta_0,...,\beta_p}{\arg\min} \sum_{i=1}^{n} \epsilon_i^2. \quad (3.2)$$

Proposition 3.1. *Denote by X the $n \times (p+1)$ matrix with each row an input vector, (with a 1 in the first position because the intercept β_0 has no weight), and similarly let y be the n-vector of outputs in the training set. If X is a full rank matrix, the optimal weights for the linear regression problem are:*

$$\hat{\beta} = (X^T X)^{-1} X^T y.$$

Proof. We can write the residual sum of squares as

$$\text{RSS}(\beta) = (y - X\beta)^T (y - X\beta).$$

This is a quadratic function in the $p + 1$ parameters. Differentiating with respect to β we obtain

$$\frac{\partial \text{RSS}}{\partial \beta} = 2X^T(y - X\beta),$$
$$\frac{\partial^2 \text{RSS}}{\partial \beta \partial \beta^T} = 2X^T X.$$

By hypothesis we know that X has full column rank, and hence $\forall\, v \in \mathbb{R}$ and $v \neq 0$, $v^T X X^T v = (X^T v)^T (X^T v) > 0$, i.e, $X^T X$ is positive definite. This result along with the fact $\text{RSS}(\beta)$ is a convex function tells us $\text{RSS}(\beta)$ admits a global minimum.

We set the first derivative to zero

$$X^T(y - X\beta) = 0,$$

and isolating β from the equation we obtain

$$\hat{\beta} = (X^T X)^{-1} X^T y.$$

□

It might happen that the columns of X are not linearly independent, so that X is not of full rank. This would occur, for example, if two of the inputs were perfectly correlated, (e.g., $x_2 = 3x_1$). Then $X^T X$ is singular and the least squares coefficients $\hat{\beta}$ are not uniquely defined. However, we will see that this is one of the assumptions made in the linear regression model.

The predicted values for an input vector x_0 are given by $\hat{f}(x_0) = (1 : x_0)^T\hat{\beta}$. The fitted values at the training inputs are $\hat{y} = X\hat{\beta} = X(X^TX^{-1})X^Ty$, where $\hat{y}_i = \hat{f}(x_i)$.

To draw inferences about the parameters and the model, we need data to meet certain assumptions. Taking these assumptions in consideration, we will be able to assess whether a simple linear regression model seems reasonable when applied to the dataset in question. Some of these data assumptions permit us to create confidence intervals for the estimated weights. According to (Molnar, 2022, Ch. 5.1)[21], these assumptions are:

- **(1) Linearity:** We assume that Equation 3.1 is the correct model for the mean; that is, the conditional expectation of Y is linear in $X_1, ..., X_p$. Hence

$$Y = \mathbb{E}(Y|X_1, ..., X_p) + \epsilon = \beta_0 + \sum_{j=1}^{p} X_j\beta_j + \epsilon. \tag{3.3}$$

  The linear regression model forces the prediction to be a linear combination of features, which is both its greatest strength and its greatest limitation. Linearity leads to interpretable models since linear effects are easy to quantify and describe. They are additive, so it is easy to separate the effects. This is one of the key reasons why the linear model, as well as other similar models, is so widely used in academic fields such as medicine or psychology.

- **(2) Normality:** We also assume that the deviations of Y around its expectation, called ϵ in Equation 3.3, are Gaussian, which means that we make errors in both negative and positive directions and make many small errors and few large errors. In other words, ϵ is a random variable following a normal distribution with expectation zero and variance σ^2, written $\epsilon \sim N(0, \sigma^2)$. If this assumption is violated, the estimated confidence intervals of the feature weights are invalid.

- **(3) Independence:** It is assumed that the observation errors are independent of each other, that is, each instance is independent of any other instance. If we perform repeated measurements, such as multiple blood tests per patient, the data points are not independent and if we use the linear regression model, we might draw wrong conclusions from the model.

- **(4) Homoscedasticity (constant variance):** The variance of the error terms σ^2 is assumed to be constant over the entire feature space. Suppose we want to predict the value of a house given the living area in square metres. We estimate a linear model which assumes that, regardless of the size of the house, the error around the predicted response has the same variance. This assumption is often violated in reality as it is plausible that the variance of error terms around the predicted price is higher for larger houses, since prices are higher and there is more room for price fluctuations. Suppose the average error (difference between predicted and actual price) in our linear regression model is 50,000€. If we assume homoscedasticity, we are assuming that the average error of 50,000€ is the same for houses that cost 1,000,000€ and for houses that cost only 40,000€. This is unreasonable because it would mean that we can expect negative house prices.

The assumptions 2, 3 and 4 can be shown mathematically as

$$C = \mathbb{E}[\epsilon\epsilon^T] = \sigma^2 I,$$

where C is the covariance matrix of observation error, I is the identity matrix, and $\mathbb{E}$ represents the expected value. In other words, the covariance matrix of the prediction error ϵ takes the following form

$$C = \begin{pmatrix} Var_{\epsilon_1} & Cov_{\epsilon_1,\epsilon_2} & \dots & Cov_{\epsilon_1,\epsilon_n} \\ Cov_{\epsilon_2,\epsilon_1} & Var_{\epsilon_2} & \dots & Cov_{\epsilon_2,\epsilon_n} \\ \vdots & \vdots & \ddots & \vdots \\ Cov_{\epsilon_n,\epsilon_1} & Cov_{\epsilon_n,\epsilon_2} & \dots & Var_{\epsilon_n} \end{pmatrix} = \begin{pmatrix} \sigma^2 & 0 & \dots & 0 \\ 0 & \sigma^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma^2 \end{pmatrix}.$$

Diagonal elements of the covariance matrix represent the variance of each observation error and they are all equal because the errors are identically distributed. The off-diagonal elements represent the covariance between two observations errors and they are all zero because the errors are independent.

- **(5) Fixed features:** The input features are considered to be "fixed", this means that they are treated as "given constants" rather than statistical variables. This means they do not have any measurement errors. This is an overoptimistic assumption. Without this assumption we would have to fit very complex measurement error models that account for the measurement errors of our input features.
- **(6) Absence of multicollinearity:** From a statistical point of view, linear regression is reasonable if the training observations (x_i, y_i) represent independent random draws from their population. We do not want strongly correlated features, because this messes up the estimation of the weights. In a situation where two features are strongly correlated, it becomes problematic to estimate the weights because the feature effects are additive and it becomes indeterminable to which of the correlated features to attribute the effects. However, if the x_i's are not drawn randomly, linear regression is still valid if the y_i's are conditionally independent given the inputs x_i.

In many cases with real data, it is difficult to satisfy all these assumptions. This does not necessarily mean we cannot use linear regression. However, if any of these assumptions is not met, the optimal performance cannot be expected and the inference of the model coefficients could be inaccurate.

3.2 Weighted linear regression

The information of this section has been extracted from (Simon and Young, 2022)[24] complemented with (Vaghefi, 2021)[25].

In this section, we look into one of the main pitfalls of linear regression: heteroscedasticity, which is the violation of the assumption 4 (homoscedasticity) introduced in the

previous section. Lack of homoscedasticity has several consequences on linear regression results. First, the performance of the models is no longer optimal. Second, model coefficients and standard errors will be inaccurate and hence their inferences and any hypothesis testing based on them will be invalid. Weighted linear regression method, also called *weighted least squares*, can be used when the ordinary least squares assumption of constant variance in the errors is violated. The model under consideration is:

$$Y = f(X) + \epsilon^* = \beta_0 + \sum_{j=1}^{p} X_j \beta_j + \epsilon^*,$$

where ϵ^* is assumed to be multivariate normally distributed with mean vector 0 and not constant variance-covariance matrix

$$C = \begin{pmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_n^2 \end{pmatrix}.$$

The diagonal elements of C are not identical and each observation has its own variance.

If we define the reciprocal of each variance σ_i^2, as the weight, $w_i = 1/\sigma_i^2$, then let matrix W be a diagonal matrix containing these weights:

$$W = \begin{pmatrix} w_1 & 0 & \dots & 0 \\ 0 & w_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & w_n \end{pmatrix}.$$

Weighted least squares is a generalisation of linear regression where the covariance matrix of errors is incorporated. Then, the weighted least squares estimate is:

$$\hat{\beta}_{WLS} = \arg\min \mathrm{WRSS}(\beta) = \underset{\beta_0,\dots,\beta_p}{\arg\min} \sum_{i=1}^{n} \boldsymbol{w_i} \left(y_i - (\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij})\right)^2 = \underset{\beta_0,\dots,\beta_p}{\arg\min} \sum_{i=1}^{n} \epsilon_i^{*2}.$$

Using the same notation we used in Proposition 3.1 it follows the analogous proposition for weighted linear regression:

Proposition 3.2. *If X is a full rank matrix, hence an invertible matrix, the optimal weights for the weighted linear regression problem are:*

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

Proof. See (Wisan, 2020)[26]. Analogous to the proof of Proposition 3.1, taking into consideration that we can write the residual sum of squares as

$$\mathrm{RSS}(\beta) = (y - X\beta)^T W (y - X\beta). \qquad \square$$

The inclusion of the W matrix in the model shows that weighted linear regression uses different weights for each observation based on their variance. A small error variance observation has a large weight since it includes more information than a large error variance observation, which has a small weight.

The main disadvantage of the weighted linear regression is that the covariance matrix of observation errors is required to find the solution. In many applications, such information is not available prior. In this case, the covariance matrix can be estimated, but it is something we will not discuss in this project.

3.3 Regularisation

This section is based on (Hastie, 2009, Ch. 3.4)[9], (Gupta, 2017)[27] and (Checka, 2018)[28] complemented with (Molnar, 2022, Ch. 5.1.7)[21] and (Phong, 2022)[29].

Avoiding overfitting is one of the most critical aspects of training our machine learning model. We have already seen in the linear regression model that we adjust the coefficients based on our training data. If our model is attempting to capture the noise in the training dataset far too hard, high coefficients will be estimated, significantly increasing the model's variance and not generalising well to future data. Noise refers to data points that are meaningless, in other words, that are not indicative of our data underlying properties. Learning such data points makes our model more flexible, but it also increases the risk of overfitting.

Regularisation tackles the overfitting problem by shrinking or regularising these learned weights to zero. In order to meet our objective, if we try to reduce the coefficients of previously learned features, the model loses accuracy. This loss in accuracy needs to be explained by something else to maintain the accuracy levels. This responsibility will be taken up by the bias portion of the model which, as seen in Section 1.6, is the part of the model that is not dependent on the training data. Regularisation attempts to generalise better in new data by reducing the model's flexibility, resulting in a drop in variance and an increase in bias.

3.3.1 Ridge regression (L2 regularisation)

When there are many correlated variables in a linear regression model, their coefficients can become poorly determined and exhibit high variance. The main premise of ridge regression is to solve this problem, so that each variable is represented appropriately relative to the contribution to the outcome. *Ridge regression* shrinks the regression coefficients by adding a penalty term to the linear regression optimisation Equation 3.2. It means that the ridge regression problem can be seen as the minimisation of two terms. Therefore, its coefficients are estimated by minimising the following penalised residual sum of squares:

$$\hat{\beta}_{ridge} = \underset{(\beta_0,\ldots,\beta_p)}{\arg\min} \left\{ \underbrace{\sum_{i=1}^{n}(y_i - (\beta_0 + \sum_{j=1}^{p}\beta_j x_{ij}))^2}_{\text{Sum of square error term (RSS)}} + \underbrace{\lambda \sum_{j=1}^{p}\beta_j^2}_{\text{Penalty term}} \right\}. \tag{3.4}$$

The term $\sum_{j=1}^{p} \beta_j^2 = \|\beta\|_2$, the L2-norm of the coefficients vector, leads to a penalisation of large weights. This type of regularisation cares a lot more about pushing down big weights than tiny ones. The "force" pushing small weights to 0 is very small.

The term $\lambda \geq 0$ is a tuning parameter[2] that decides how much we want to penalise the flexibility of our model. The increase in flexibility of a model is represented by an increase in its coefficients, and if we want to minimise the above function, then these coefficients need to be small. This is how the ridge regression technique prevents coefficients from rising too high. When $\lambda = 0$, the penalty term has no effect, and the estimates produced by ridge regression will be equal to the ones found in least squares. However, as $\lambda \to \infty$, the impact of the shrinkage penalty grows, and the ridge regression coefficient estimates will approach zero. As it can be seen, selecting a good value of λ is critical. Cross-validation comes in handy for this purpose.

The value of λ should be carefully selected. Until a point, its increase is beneficial as it is only reducing the variance (hence avoiding overfitting), without losing any important properties in the data. But after a certain value, the model starts losing important properties, giving rise to bias in the model and thus underfitting.

The coefficients that are produced by the standard least squares method are scale equivariant, i.e., if we multiply each input by c then the corresponding coefficients are scaled by a factor of $1/c$. Therefore, regardless of how the predictor is scaled, the multiplication of coefficient and predictor ($\beta_j X_j$) remains the same. However, this is not the case with ridge regression, and therefore, we need to standardise the predictors or bring the predictors to the same scale before solving Equation 3.4. The formula used to do this is given below:

$$\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)}}.$$

In addition, notice that the intercept β_0 has been left out of the penalty term. Its penalisation would make the procedure become dependent on the original choice of the target value vector Y. Adding a constant c to each target will not result in the same shift in the prediction for y. Thus, it is better to leave the intercept out of the penalisation.

The remaining coefficients get estimated by a ridge regression without intercept, using the centered $\tilde{x}_{ij}$. Henceforth we assume that this centering has been done, so that the input matrix X has p (rather than $p+1$) columns.

Writing the Equation 3.4 in matrix form,

$$RSS_\lambda(\beta) = (y - X\beta)^T(y - X\beta) + \lambda\beta^T\beta,$$

it can be proved that the ridge regression solutions are

$$\beta_{ridge} = (X^T X + \lambda I)^{-1} X^T y,$$

[2] If λ was not a hyperparameter, gradient descent would nicely set it to 0 and travel to the global minimum. Hence, the control on λ cannot be given to gradient descent and needs to be kept out.

where I is the $p \times p$ identity matrix. Notice that with the choice of quadratic penalty $\beta^T\beta$, the ridge regression solution is again a linear function of y. The solution adds a positive constant to the diagonal of X^TX before inversion. This makes the problem nonsingular even if X^TX is not of full rank, and was the main motivation for ridge regression when it was first introduced in (Hoerl and Kennard, 1970)[30].

3.3.2 Lasso regression (L1 regularisation)

In reality we might not have just a handful of features, but hundreds or thousands. And our linear regression model interpretability goes downhill. We might even find ourselves in a situation where there are more features than instances, and we cannot fit a standard linear model at all. This disadvantage is overcome by *lasso regression*, which introduces *sparsity* to the model, i.e., few features. Instead of squaring the coefficients like in ridge regression's L2 penalty term, lasso utilises the L1 penalty[3], taking the coefficients absolute value and estimating them by minimising the following penalised residual sum of squares:

$$\hat{\beta}_{lasso} = \underset{(\beta_0,\ldots,\beta_p)\in\mathbb{R}^n}{\arg\min} \left\{ \underbrace{\sum_{i=1}^{n}(y_i - (\beta_0 + \sum_{j=1}^{p}\beta_j x_{ij}))^2}_{\text{Sum of square error term (RSS)}} + \underbrace{\lambda \sum_{j=1}^{p}|\beta_j|}_{\text{Penalty term}} \right\}. \tag{3.5}$$

The term $\sum_{j=1}^{p}|\beta_j| = \|\beta\|_1$, the L1-norm of the feature vector, does not penalise big weights as much as ridge regression, but it makes large and small weights a bit smaller. In comparison to ridge, it tends far more to drive small weights to 0.

The solution for lasso regression, $\hat{\beta}_{lasso}$, is not linear in the y_i. The nature of absolute value makes $\hat{\beta}_{lasso}$ have no closed form expression as $\hat{\beta}_{ridge}$. Computing the lasso solution is a quadratic programming problem, although efficient algorithms are available for computing the entire path of solutions as λ is varied, with the same computational cost as for ridge regression. Lasso, as a feature selection technique, will also converge faster to the final solution than other feature selection techniques such as *best subset*, *forward stepwise* or *backward stepwise*, which may be less efficient in terms of computation time.

3.4 Graphical representation of ridge and lasso regression

Let us take a look at the above methods with a different perspective. The ridge regression can be thought of as solving an equation, where summation of squares of coefficients is less than or equal to C. Therefore, we can reformulate ridge regression as

$$\hat{\beta}_{ridge} = \underset{\beta_0,\ldots,\beta_p}{\arg\min} \left\{ \sum_{i=1}^{n}(y_i - (\beta_0 + \sum_{j=1}^{p}\beta_j x_{ij}))^2 \right\} \text{ subject to } \sum_{j=1}^{p}\beta_j^2 \leq C. \tag{3.6}$$

[3] We use L1 and L2 norms as we need to generalise in the entire coordinate space, i.e., positive and negative values on axes.

And the lasso can be thought of as an equation where summation of modulus of coefficients is less than or equal to C. Therefore, we can reformulate lasso as

$$\hat{\beta}_{lasso} = \underset{\beta_0,\ldots,\beta_p}{\arg\min} \left\{ \sum_{i=1}^{n} (y_i - (\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}))^2 \right\} \text{ subject to } \sum_{j=1}^{p} |\beta_j| \leq C. \tag{3.7}$$

In order to keep things simple and provide a visual representation, suppose we just have two weights, β_1 and β_2. Then, these last two equations can be seen graphically as:

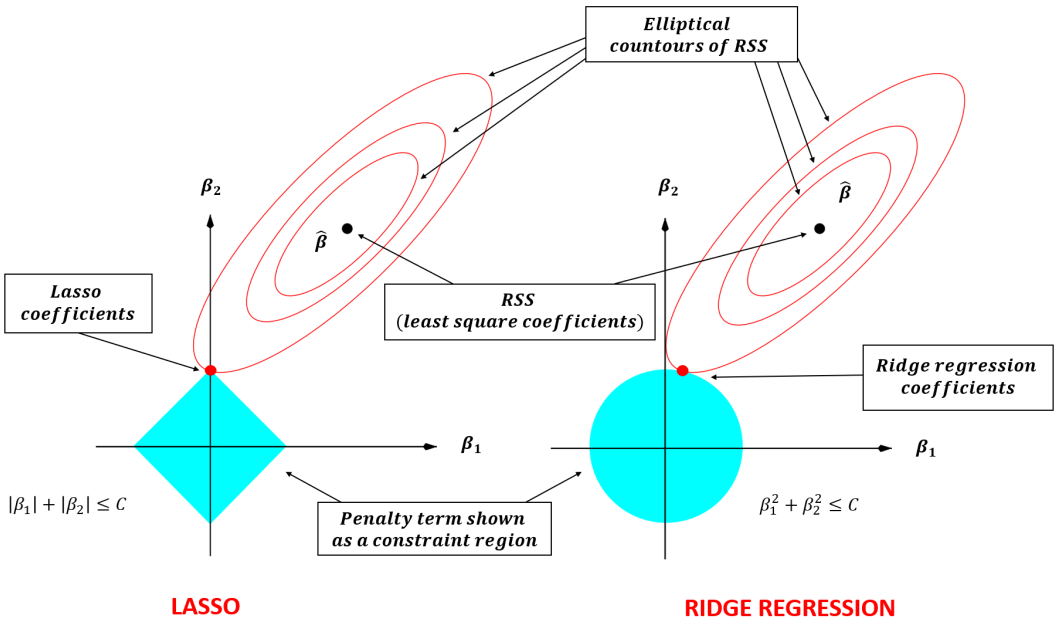


FIGURE 4: Estimation picture for the lasso and ridge regression. Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions $|\beta_1| + |\beta_2| \leq C$ and $\beta_1^2 + \beta_2^2 \leq C$, respectively, while the red ellipses are the contours of the least squares error function.

Figure 4 shows the constraint regions given (blue areas), for lasso and ridge regression, along with contours for RSS (red ellipses). The ellipses are the contour of the gradient descent[4] algorithm for the RSS function, which is projected in 2 dimensions *(See Appendix D)*.

There is a one-to-one correspondence between C and the complexity parameter λ, as C increases, λ decreases, and vice versa. For C big enough, the blue regions will contain the centre of the ellipse, making coefficient estimates of both regression techniques, equal to the least squares estimates ($\hat{\beta}$), indicating that the regularisation term λ is equal to 0. Contrarily, as λ increases, the flexibility of the model decreases, leading to increased bias but decreased variance. This means that the cost function for ridge regression will be more and more dependent on the magnitude of the regularisation term, forcing the coefficients to be relatively small, closer to 0. The fact that these coefficients are relatively small suggests that C is also small. So, as λ increases, C decreases.

[4] Gradient descent is an iterative first-order optimisation algorithm used to find a local minimum of a given function. In our case the RSS cost/loss function, which is a convex function, so in this case we will converge to a global minimum.

In Figure 4, C is not big enough for the blue region to contain $\hat{\beta}$. In this case, the lasso and ridge regression coefficient estimates are given by the first point at which an ellipse contacts the constraint region. Since ridge regression has a circular constraint with no sharp points, this intersection will not generally occur on an axis, and so the ridge regression coefficient estimates will be exclusively non-zero. However, the lasso constraint has corners at each of the axes, and so the ellipse will often intersect the constraint region at an axis, setting one of the coefficients equal to zero. In higher dimensions, where we have more than two parameters, many of the coefficient estimates may equal zero simultaneously *(See Appendix E)*.

3.5 What is explainable artificial intelligence?

A *black box* is a system which can be viewed in terms of its inputs and outputs, without revealing its internal mechanisms. Its implementation is "opaque" (black). In machine learning, black box describes models that cannot be understood by looking at their parameters. However, when making life-changing decisions like a medical diagnosis, it's crucial to understand why we are making that decision. The importance of describing AI outputs becomes clear at this point. Problematically, despite their apparent power in terms of the results and predictions, many machine learning algorithms suffer from opacity, making it impossible to get insight into their internal mechanisms. This is a critical issue, as trusting key decisions from a system that is unable to explain itself has clear risks. *Explainable artificial intelligence (XAI)* suggests a transition towards a more transparent AI to overcome this problem. Its goal is to develop a set of techniques that either produce more understandable models keeping high levels of performance or provide external tools to better understand the models that are inherently not interpretable.

3.6 XAI taxonomy

This section is based on (Angelov, 2021)[31] complemented with (Molnar, 2022, Ch.3)[21].

A number of terminologies exist in the literature to express the opposite of a black box model. The following terms are distinguished:

- ***Transparency***: A model is considered to be transparent if, by itself, it has the potential to be understandable. In other words, transparency is the opposite of black box.
- ***Interpretability***: Capacity to provide interpretations in terms that are understandable to a human.

  A model is more interpretable than another if its decisions are simpler for humans to understand than the other model's decisions.

  There is no mathematical definition of interpretability in the field of explainable artificial intelligence, as the degree of interpretation is not an exact concept. An approach to a definition from a mathematical point of view is given in (Tjoa, 2020)[32], where it defines interpretability via a mathematical structure.

Also mention that the notion of interpretability depends on the target audience. ML experts may be able to interpret a complex model, but a doctor may be more comfortable with a small number of weighted features as an explanation.

- ***Explainability***: It is related with the notion of explanation as an interface between humans and an AI system. We will use thet erm *Explanation* for explanations of individual predictions.

If hundreds or thousands of features significantly contribute to a prediction, it is not reasonable to expect any user to comprehend why the prediction was made, even if individual weights can be inspected. This requirement further implies that explanations should be easy to understand, which is not necessarily true of the features used by the model, and thus the "input variables" in the explanations may need to be different than the features. As we see, interpretability is insufficient as it does not address all the issues that might arise when dealing with black box models. Explainability, rather than basic interpretability, is essential to earn user trust and get significant insights into the causes, reasons, and decisions of black box techniques. Although by default, explainable models are interpretable, the opposite is not always true.

Methods for machine learning interpretability can be classified according to various criteria as in the following image:

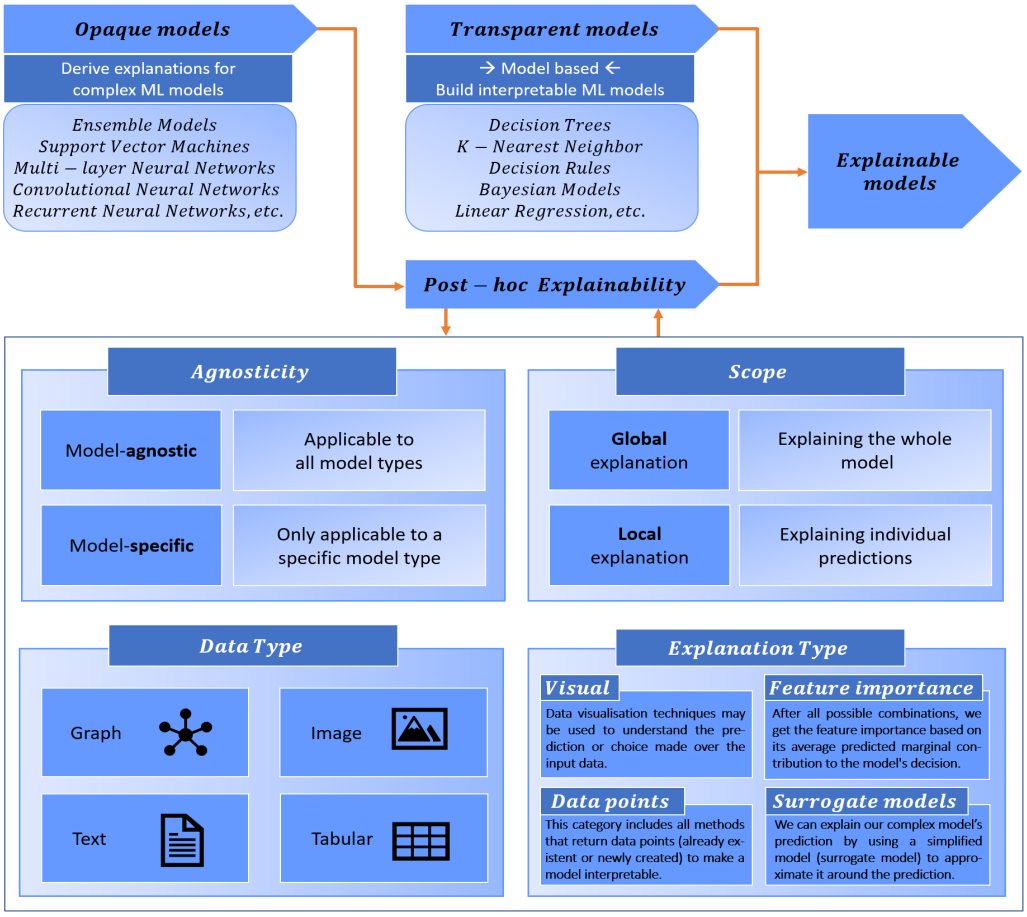


FIGURE 5: General ontology of explainable artificial intelligence. Picture inspired by (Angelov, 2021)[31].

3.7 LIME

The information in this section has been extracted from the paper where the LIME was presented, (Riberio, 2016)[33], complemented with (Molnar, 2022, Ch. 9.1)[21], (Riberio, 2016)[34], (DeepFindr, 2021)[35] and (Sharma, 2018)[36].

LIME stands for Local Interpretable Model-agnostic Explanations and it is an algorithm that can explain individual predictions of any classifier or regressor in a faithful way.

Each part of the LIME's name reflects something that we desire in explanations. **Local** refers to *local fidelity*, i.e., we want the explanation to really reflect the behaviour of the classifier in the neighbourhood of the instance being predicted. This explanation is useless unless it is **interpretable**, that is, unless a human can make sense of it. LIME can be applied to any machine learning model, so it is **model-agnostic**.

3.7.1 Data representation

As shown in Figure 5, different forms of data may be used to train our machine learning model, therefore, it is important to distinguish between features and interpretable data representations. As mentioned before in Section 3.6, interpretable explanations need to use a representation that is understandable to humans, regardless of the actual features used by the model. For example, a possible interpretable representation for text classification is a binary vector indicating the presence or absence of a word, even though the classifier may use incomprehensible features such as *word embeddings*. Likewise for image classification, an interpretable representation may be a binary vector indicating the presence or absence of a *super-pixel*[5], while the classifier may represent the image as a *tensor*[6]. These interpretable representations allow LIME to operate for different data types.

Formally, if we denote $x \in \mathbb{R}^d$ as the original representation of an instance being explained, LIME uses simplified inputs x' that map to the original inputs through a mapping function $x = h_x(x')$. The simplified input $x' \in \{0,1\}^{d'}$ is interpreted as a binary vector for its interpretable representation. This means that we transform our original feature vector into a discrete binary vector where features are included or excluded. The way h is defined allows us to recover x' from our original input x through the inverse of h, that is, $x' = h_x^{-1}(x)$. Note that $x = h_x(x')$ even though x' may contain less information than x because h_x is specific to the current input x.

Given this disjunctive between features and interpretable data representations, we now can assume we are dealing with interpretable features like the age of a person. Moreover, as the title of this project denotes, in Section 3.11 we will use XAI to get insights in a diabetes prediction problem. Because the diabetes database is in tabular format, we will also assume we are working with this type of data. As a result of these assumptions, an instance x is itself an interpretable representation, that is, $x = x'$. Thus, for the sake of notation simplicity, LIME will be presented without distinguishing between x and x'.

[5] Super-pixel: Contiguous patch of similar pixels.

[6] Tensor: Mathematical objects that generalise scalars, vectors and matrices to higher dimensions.

3.7.2 LIME overview

Before diving into the mathematics, we will make a brief overview of LIME using an abstract graphical approach:

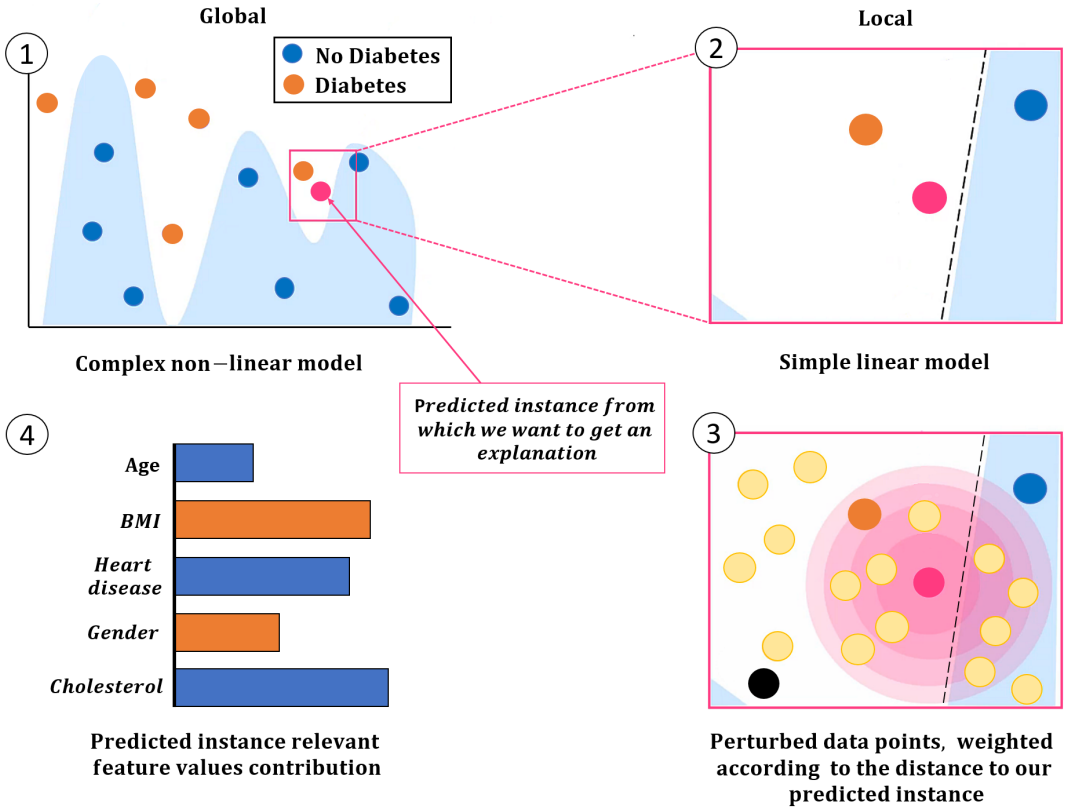


FIGURE 6: LIME general pipeline for explaining individual predictions of a classifier trying to determine if a patient has diabetes or not.

Looking at Figure 6, we see that LIME can be divided into four steps:

(1) The decision boundary of our original black box model is represented by the blue/white background, and as we can see, it is highly non-linear. The pink point is the instance we want to be explained, and it has been predicted as "Diabetes". How can we explain that our model predicts diabetes without peaking into the complex model we receive our predictions from? We cannot reduce the whole limit boundary in a unique explanation, that is a global explanation. LIME's main idea is to zoom in a neighbourhood of the prediction which leads us to step 2.

(2) Now we can do a simpler explanation of this local region, this way we do not worry about the rest of the model and we obtain an explanation equally valid. It is worth mentioning that features that may be important in a local context may not be globally important and vice versa. In other words, a local explanation does not imply a global explanation. What LIME does is to present a local explanation through a transparent surrogate model, in this case a linear model in the region surrounding our prediction. But how do we train this linear model? This leads us to step 3.

(3) We simply perturb instances (yellow points) around our pink point and weight them according to their proximity to it. We get the original model's prediction on these perturbed instances, and then learn a linear model (black line) that approximates the model well in the vicinity of our instance. Note that classifying the black point incorrectly in the surrogate model is not relevant since it is far away from our pink point.

(4) Finally, in this step we receive our explanation by interpreting the local model. The bar charts represent the importance given to the most relevant features, i.e., the features that have most contributed to the prediction of our pink point. In the case of using a linear regression as a surrogate model, these values would be the fitted weights of the model.

3.7.3 LIME mathematical optimisation problem

Fidelity-Interpretability Tradeoff

Let us first introduce the necessary elements to understand the LIME optimisation problem:

- f: Complex model being explained. In classification $f(x)$ is the probability (or a binary indicator) that x belongs to a certain class. For multiple classes, we explain each class separately, thus $f(x)$ is the prediction of the relevant class.
- g: Surrogate model we use to approximate f in the vicinity of x.
- G: Family of possible interpretable models. For example, linear models such as lasso, decision trees, or decision rules.
- π_x: Local neighbourhood of x.
- $\mathcal{L}(f, g, \pi_x)$: Indicates a measure of how *unfaithful* g approximates f in a neighbourhood defined by π_x.
- $\Omega(g)$: Complexity measure of the model g that is used to regulate the complexity of the surrogate model. In the case of decision trees it can refer to the depth of the tree or in a linear regression, the number of null weights. Here we assume complexity is opposed to explainability.

Once introduced this notation, the explanation produced by LIME at a local point x is obtained by the following generic formula:

$$\xi(x) = \underset{g \in G}{\arg\min}\, \mathcal{L}(f, g, \pi_x) + \Omega(g). \tag{3.8}$$

The objective is to assure interpretability and local fidelity while minimising $\mathcal{L}(f, g, \pi_x)$ maintaining $\Omega(g)$ small enough to be interpretable for a human.

In summary, our loss function $\xi(x)$ tells us to find a simple model $g \in G$ where the model fidelity is maximised in the neighbourhood π_x of our instance, keeping the interpretability as simple as possible.

Sampling for Local Exploration

We want to minimise the local loss $\mathcal{L}(f, g, \pi_x)$ without making any assumptions about f, since we want the explainer to be model-agnostic. Thus, in order to learn the local behaviour of f as the inputs vary, we approximate $\mathcal{L}(f, g, \pi_x)$ by generating perturbed instances, weighted by π_x. How do we generate these perturbations? In tabular data, LIME creates new instances individually perturbing each feature of x from a normal distribution inferred from the training set. Given this dataset $\mathcal{Z}$ of perturbed samples with the associated labels, we optimise Equation 3.8 to get an explanation $\xi(x)$.

Sparse linear explanations

LIME chooses the family of surrogate interpretable models G as the class of linear models ($g(z) = w_g \cdot z$). This type of model is used to minimise a weighted linear regression:

$$\mathcal{L}(f, g, \pi_x) = \sum_{z \in \mathcal{Z}} \pi_x(z)(f(z) - g(z))^2.$$

We minimise the squared distances between the predictions of our complex model f and our simpler model g. This is done for all perturbed instances assigning each of them a weight $\pi_x(z)$.

Let $\pi_x(z) = \exp(-D(x, z)^2/\sigma^2)$ be an *exponential kernel* defined on some distance function D, usually the euclidean distance, and *kernel width* σ^2. The exponential kernel attributes a value in the range $[0, 1]$, the higher the closer to the reference point, while the kernel width σ^2 decides how large the circle of the meaningful weights around it is.

It is worth noting that this method is fairly robust to sampling noise since the samples are weighted by π_x. Also, since LIME produces an explanation for an individual prediction, its complexity does not depend on the size of the dataset, but instead, on time to compute $f(x)$ and on the number of perturbed samples N.

We ensure that the explanation is interpretable for tabular data by predefining a set of features K that we wish to have in our interpretable model. Although a greater K potentially results in more fidelity in the model, the lower K, the simpler the model is to understand. For training models with exactly K features, lasso, which we heave already introduced in Subsection 3.3.2, is a convenient option. Why? A lasso model with a high regularisation parameter λ produces a featureless model. By retraining the lasso model with slowly decreasing λ, we can exactly get K weights that differ from zero, thus obtaining K features. As LIME uses lasso as sparse linear model, this particular choice of Ω directly solves Equation 3.8.

LIME's biggest advantage is its agnosticity. Even if the underlying machine learning model is replaced, the same local interpretable model can be used for explanation. Another advantage of LIME is that the generated explanations are brief and understandable to humans. As a result, the LIME application may be more appropriate in situations when the recipient of the explanation is a layperson or someone with limited time. In situations when we may be legally obligated to explain a prediction thoroughly, this is insufficient.

3.7.4 LIME limitations

- **Neighbourhood:** When using LIME with tabular data, the right definition of the neighbourhood is a huge, unsolved challenge. According to (Molnar, 2022)[21], this is LIME's most serious flaw, and the reason why it should be used only with extreme caution *(See Appendix F)*.
- **Non-linearity:** Because we selected G as the family of sparse linear models, there may not be a faithful explanation if the underlying model is very non-linear even in the prediction's locality.
- **Improbable instances:** Perturbed instances are produced from a Gaussian distribution without regard for feature correlation. This can result in improbable instances that can be used to learn local explanation models.
- **Instability:** In (Alvarez, 2018)[37], the authors showed that in a simulated situation, the explanations for two relatively close points changed substantially. Furthermore, if one repeat the sample procedure, the results might be different. Because instability makes it harder to believe explanations, one should be very critical.

3.8 Submodular Pick-LIME (SP-LIME)

The information in this section has been extracted from (Riberio, 2016)[33], (Sharma, 2018)[36] and (Knor, 2021)[38].

Although the explanation of a single instance's prediction gives us some credibility of the model, it is insufficient to evaluate the model's overall interpretability. SP-LIME proposes to give a global explanation of the model by explaining a set of representative individual instances.

SP-LIME algorithm is still model-agnostic as it runs LIME for all available/selected instances. According to LIME's author, "even though explanations of many instances might be insightful, these instances need to be selected judiciously, since consumers may not have the time to review a large number of explanations" (Ribeiro, 2016)[33].

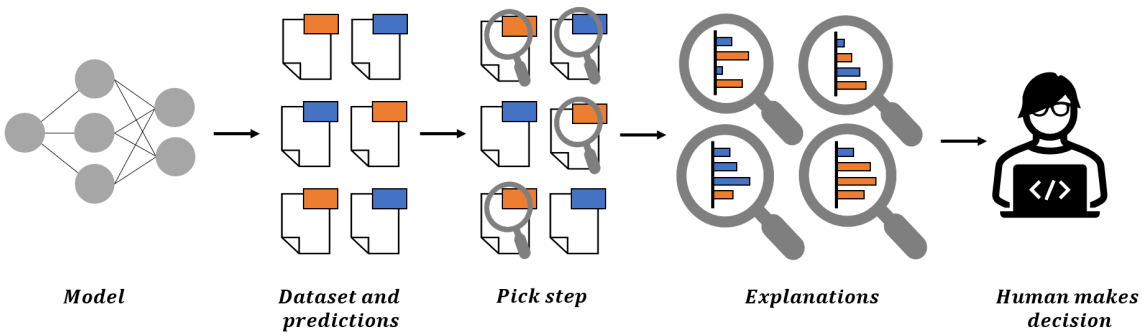


FIGURE 7: Decision-making diagram process using SP-LIME as complement to a machine learning model. First, the model is trained and the predictions are obtained. We next utilise SP-LIME to generate a set of model-representative explanations. As a result, the model explanation will be used in conjunction with the model's prediction to help the user make the final decision.

Algorithm prerequisites

- B: As recently mentioned, the LIME's author argues that the user may not have time to examine many different explanations. Then the number of explanations (instances) that one is willing to look at to understand the model is represented as a *budget* with the letter B.

- W: Given the explanations of the original database X, we construct an $n \times p$ matrix, which we will denote W, where n is the number of instances and p the number of features. The *explanation matrix* W represents the local importance of the features for each instance, i.e, the lasso regression weights obtained from the surrogate linear model fitted around each instance.

- I: We will call I_j the global importance of the feature (column) j in the matrix W. We want variables that explain many different instances to have a higher value.

  I_j is defined as $I_j = \sqrt{\sum_{i=1}^{n} |w_{g_{ij}}|}$ where $w_{g_{ij}}$ is the assigned weight to the feature j for the explanation of instance i in our linear surrogate model g.

  In Figure 8 we can see an example where $n = p = 5$ and W is binary ($K = 2$ for lasso). Because feature f_2 is used to explain more instances, the importance function I should score it higher than feature f_1, i.e. $I_2 > I_1$. While we want to choose instances that cover the most essential components, the set of explanations must not be redundant in the components they present the users, thus avoid picking instances with similar explanations. Figure 8 also shows that after picking the second row, the third row does not provide new information because the user has already seen features f_2 and f_3, however, the fourth row introduces the user to wholly new features. All of the cells are totally covered when the second and last rows are picked.

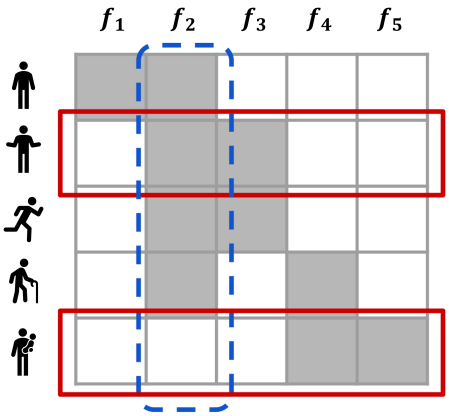


FIGURE 8: Rows represent instances (patients) and columns represent features (patient characteristics). Feature f_2 (dotted blue) has the highest importance since it is the one that explains more instances. The pick procedure would select rows 2 and 5 (in red), covering all features except f_1. Figure adapted from (Ribeiro, 2016)[33].

- $c(V, W, I)$: This non-redundant coverage notion is formalised as follows:

$$c(V, W, I) = \sum_{j=1}^{p} \mathbb{1}_{[\exists\, i \in V:\, |W_{g_{ij}}| > 0]} I_j.$$

  This is the *coverage function* c, and given W and I, it calculates the total importance of the features that appear in at least one instance in a set V. As we can see in the indicator function, I_j will be summed to the coverage if there exists one instance in the set V that its assigned weight to the variable j for the instance i is greater than 0.

Algorithm

The following algorithm selects the B instances with a highest coverage:

Algorithm 1 Submodular pick (SP) algorithm

Require: Instances X, Budget B
– (1)
for all $x_i \in X$ **do**
  $W_i \leftarrow$ **explain**(x_i) ▷ Using LIME
end for
– (2)
for $j \in \{1, \ldots, p\}$ **do**
  $I_j = \sqrt{\sum_{i=1}^{n} |w_{g_{ij}}|}$ ▷ Compute feature importances
end for
– (3)
while $|V| < B$ **do** ▷ Greedy optimisation of $Pick(W, I) = \underset{V, |V| \leq B}{\arg\max}\ c(V, W, I)$
  $V \leftarrow V \cup \arg\max_i\ C(V \cup i, W, I)$
end while
return V

As we can see the algorithm is divided in 3 steps:

(1) Applies LIME to all the instances of the original dataset X.

(2) For each column (feature) it calculates its importance.

(3) This is called the *pick-step*, and it is the task of selecting B instances for the user to inspect. Initialises a set V to empty and while the cardinal of V is less than the budget B, it iteratively adds the instance that maximises the coverage function. Finally returns the set V, which is the set of B instances that better explain the complex model.

3.9 SHAP

The information in this section has been extracted from the paper where SHAP was presented (Lundberg and Lee, 2017)[39], (Molnar, 2022)[21] and (Sahakya, 2021)[40], complemented with (Machine learning TV, 2021)[41], (DeepFindr, 2021)[42] and (Kie, 2021)[43].

3.9.1 Shapley Values

Let us imagine we have a group of different people cooperating in a game, where the cooperative game could be a machine learning competition. After the game is over, the winning group receives a prize, for example 10,000€. The question would be the following: How would the money be distributed among all the members of the group so that the distribution is fair? Well, there are members who have contributed more and distributing it equally would not be fair. The answer to this question is given by the *Shapley values*, introduced in (Shapley, 1951)[44].

Before describing how Shapley values are calculated, we will present them in a more formal setting with some important notation. A *coalitional game* typically consists of a set N of $|N| = n$ players and a *characteristic function* $v : 2^n \longrightarrow \mathbb{R}$ with $v(\emptyset) = 0$ that specifies the value of each possible *coalition* (subset of players). For a coalition S, this value $v(S)$ describes the contribution of S to the output. The *grand coalition* is the one consisting of all the participants. The *marginal contribution* of a player p_i to a coalition S_j is the difference in value that p_i produces upon joining S_j. The Shapley value is based on the marginal contribution concept, but it takes into account all alternative sequences in which the participants may have joined the grand coalition. Intuitively, the Shapley value attempts to fairly measure the contribution of each player to the coalition consisting of all players.

For example, given four players $\{p_1; p_2; p_3; p_4\}$, one of the possible joining orders is (p_2, p_4, p_1, p_3), which indicates that p_2 was the first to join the grand coalition, followed by p_4, p_1, and lastly p_3. For example, when estimating the Shapley value of p_4, we analyse all possible joining orders and compute the marginal contribution of p_4 to the coalition of players who joined before p_4 for each such order. Then, the Shapley value of p_4 is simply its average marginal contribution, taken over all possible joining orders.

A 3-player example is illustrated in the following figure:

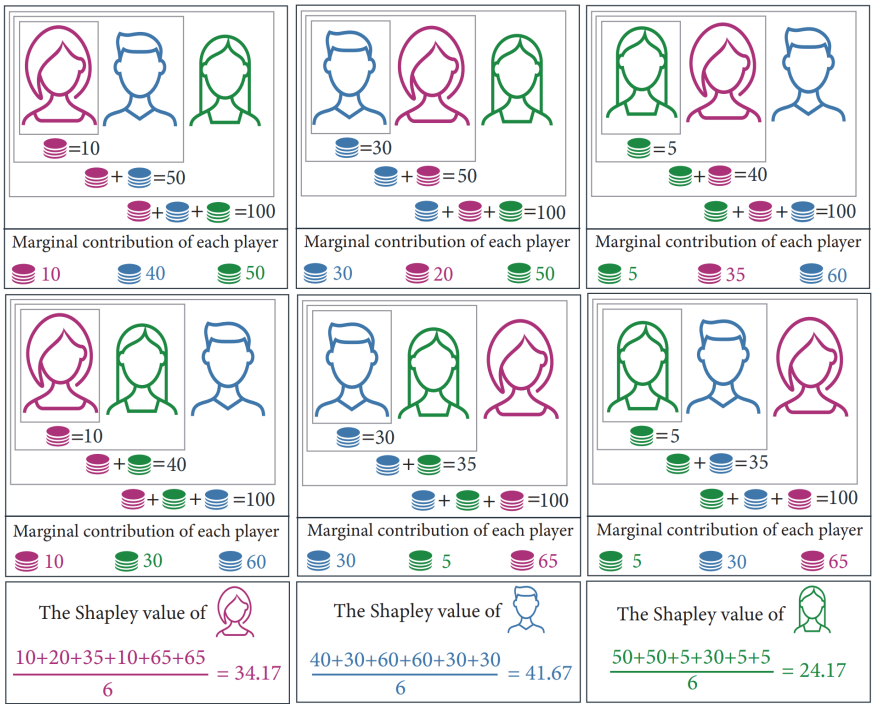


FIGURE 9: An illustration of the Shapley value. Given three players, namely Red, Blue and Green, one possible joining order is that Red joined first, followed by Blue, then Green (see the top-left corner). Another possible joining order is that Green joined first, followed by Red, then Blue (top-right corner). For every joining order, the Shapley value computes the marginal contribution of each player, which is the difference in value caused when that player joins the coalition. For example, given the joining order in the top-left corner, the marginal contribution of Red, Blue and Green is 10, 40, and 50, respectively. Then, the Shapley value of each player is defined as its average marginal contribution, taken over all joining orders (see the bottom row of the figure). Image adopted from (Sahakyan, 2021) [40].

Shapley value formula

According to the (Shapley, 1951)[44] the amount that player i is given in a coalitional game (v, N) is

$$\phi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!}(v(S \cup \{i\}) - v(S)). \tag{3.9}$$

The terms of the formula can be interpreted in the following way:

- $v(S \cup \{i\}) - v(S)$: Captures the marginal contributions of member i.
- $|S|!$: Ways in which the set S could have been formed before adding i.
- $(|N| - |S| - 1)!$: Different ways the remaining players could be added.
- $|N|!$: Number of combinations that can be formed with the coalition.

The last three terms assign to the marginal contribution an specific weight accounting for all the different sequences in which the total coalition can be formed. The idea is that small and large subsets are more important, thus they get a higher weight. For subsets with many players, we can learn about this player's total effect (main effect plus feature interactions). For small subsets we have isolated players and we can directly observe its is contribution to the payout. Contrarily, if a coalition consists of half the players, we learn little about an individual feature's contribution, as there are many possible coalitions with half of the players. (*For an example of Shapley value computation see Appendix G*).

3.9.2 SHAP framework

Translating the concept of Shapley values into the field of model explainability is relatively straightforward and this is exactly what Scott Lundberg and Su-In Lee did in 2017 with the paper "A unified approach to interpreting model predictions", where they introduced SHAP (Lundberg and Lee, 2017)[39].

SHAP stands for SHapley Additive exPlanations and it reframes the problem of Shapley values from a problem where we look at how the members of a coalition contribute to generating a result V to another where we look at how individual features contribute to the output of a model. In other words, the coalition game would be the machine learning model, and the payout would be a prediction.

They do it in a very specific way, which we can get an idea of by looking at the model name. We already know what Shapley values are, we also know what an explanation is, but what about "additive"? This part refers to additive feature attribution methods.

Additive feature attribution methods

A model's best explanation is the model itself; it properly reflects itself and it is straightforward to comprehend. We cannot use the original model as the best explanation for sophisticated models like ensemble techniques or deep networks since they are difficult to understand. We must instead take advantage of a simplified *explanation model*, which is defined as any interpretable approximation to the original model.

Unlike before with LIME and because we believe it is essential to comprehend the entire SHAP structure, now we will distinguish between an input x and its interpretable representation x' (for tabular data, this notation is not that important). As we already commented in Subsection 3.7.1, it means that we transform a vector of variables into a binary discrete vector where the variables are included or excluded. Then we can define the additive feature attribution method as the explanatory model g satisfying the following two properties:

(1) This is the property local methods try to ensure and states that:

$$if\ x \approx x'\ then\ f(x) \approx g(x').$$

(2) The complex model g must take the following form:

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

where ϕ_i tells us how much a feature changes the model's output, which we call *attribution*, and p is the number of features of the model.

If we have these two properties for the model, we have a model that has additive attribution of the features. The advantage of this type of explanation is that it is very easy to interpret because we can see the exact contribution of each feature to the prediction of an instance just by looking at the ϕ_i values.

Properties that uniquely determine additive feature attribution methods

(1) Local accuracy

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

The prediction of the explanatory model $g(x')$ is the same as the original model $f(x)$ when $x = h_x(x')$.

(2) Missingness

$$x'_i = 0 \implies \phi_i = 0.$$

When a feature is left out of the model, its attribution must be 0. That is, only the presence of features, not their exclusion, may change the output of the explanation model.

(3) Consistency

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

$$\forall\ z' \in \{0,1\}^p,\ f'_x(z') - f'_x(z' \backslash i) \geq f_x(z') - f_x(z' \backslash i) \implies \phi_i(f', x) \geq \phi_i(f, x).$$

If the original model varies in such a way that the contribution of a feature varies, the attribution in the explanatory model cannot vary in the opposite direction. For example, if we have a new model where a specific feature has a higher contribution in the original model, the attribution in our new explanation model cannot be reduced.

SHAP authors state that these qualities are well-known in Shapley value estimate methods, but they were previously unknown in additive feature attribution approaches. These 3 properties are defined as *efficiency*, *symmetry*, *dummy* and *additivity* in (Molnar, 2022)[21], however, it can be proved that these two definitions are equivalent.[7]

One surprising characteristic is that there is only one attribute class in which an attribute method satisfies these three desirable properties. This can be formalised with the following theorem:

Theorem 3.3. *Only one possible explanation model g follows additive feature attribution methods definition and satisfies Properties 1,2, and 3:*

$$\phi_i(f,x) = \sum_{z' \subseteq x'} \frac{|z'|!(p-|z'|-1)!}{p!}(f_x(z') - f_x(z' \backslash i)), \tag{3.10}$$

where $|z'|$ is the number of non-zero entries in z', and $z' \subseteq x'$ represents all z' vectors where the non-zero entries are a subset of the non-zero entries in x'.

Proof. See (Shalpey, 1953, pp. 301-317)[46]. □

Theorem 3.3 shows that there is only one possible additive feature attribution method satisfying the three properties, the one taking as attributes the Shapley values. This result implies that feature attribution methods not based on Shapley values violate at least one of these 3 properties. LIME is one of these feature attribution methods which, as it is not based on Shapley values , does not satisfy the three properties. It satisfies Property 1 and Property 2 but not Property 3.

In Equation 3.10, the Shapley value formula in a machine learning context is introduced. How do we interpret it? The value $\phi_i(f,x)$ gives us the Shapley value (contribution) for feature i in a model f. Note that $\phi_i(f,x)$ gives us the Shapley value for the specific value that feature i takes in the input data point x, not to the feature in a global sense.

This data point x could be a row in a tabular dataset. The first thing we see in the formula is the summation for all subsets Z' of our interpretable input x', that is, all the feature combinations, so we take into account the interactions between them. The weighting is exactly as we saw in Equation 3.9. In the last part of the formula we obtain the output of the model with and without the feature i. The difference between these two values tells us how much has i contributed in the prediction of the subset z'. This part is the one corresponding to the marginal contribution of a player in Equation 3.9, but now for a feature i. Then we do this for all subsets z' and we obtain the contribution of each feature to the prediction of the instance x.

Machine learning models have inputs of a fixed size and we cannot change their shape. Then the following question may arise: How do we exclude a variable in a machine learning model? The way this is solved is simple, the missing variables are imputed by random values of this variable from our train dataset.

[7] See (Molnar, 2019)[45].

The Shapley value takes a long time to compute. Only the approximate solution is feasible in the majority of real-world problems. Computing the exact Shapley value is computationally costly, there are 2^n possible subsets of the feature values, where M is the number of features. Only 64 subsets must be calculated for a set with four variables; however, with 32 variables, more than 17 billion subsets must be calculated, which is algorithmically too expensive. So to get around this problem, Lundberg and Lee devised the *Kernel SHAP*, a way of approximating Shapley values through far fewer subsets.

3.9.3 Kernel SHAP (Linear LIME + Shapley values)

Kernel SHAP is a model-agnostic framework for outcome explanation, which combines LIME with the Shapley value. It estimates for an instance x the contributions of each feature value to the prediction. Kernel SHAP consists of four steps:

(1) Sample coalitions $z'_k \in \{0,1\}^M$, where $k \in \{1,\dots,K\}$ is the number of samplings. ($1 =$ feature present in coalition, $0 =$ feature absent).

(2) Get prediction for each z'_k by first converting z'_k to the original feature space and then applying model f. As said before, the machine learning model does not let us omit a feature. So what we do is define a background dataset B, using instances on which the model has been trained. What it does is to fill our omitted feature(s) with values from B, keeping the original values of the non-omitted features.

(3) When we have already calculated the K subsets, we can formulate it as a linear regression weighted with a coefficient assigned to each variable, the *Shapley Kernel* weight introduced in Theorem 3.3.

(4) Return Shapley values ϕ_i, the coefficients from the weighted linear regression fitted with the sampled subsets z'_k.

Linear LIME approximates f locally using a linear explanation model. At first look, LIME's regression formulation in Equation 3.8 appears to be rather different from Shapley value formulation introduced in Equation 3.10. We know the Shapley values are the only viable solution to Equation 3.8 that satisfies properties 1-3, since LIME is an additive feature attribution method. A question that may arise is whether the solution to Equation 3.8 recovers Shapley values. The solution is determined by the loss function $\mathcal{L}$, the weighting kernel $\pi_{x'}$, and the regularisation term Ω. The LIME choices for these parameters are made heuristically and using these choices, as a result, Equation 3.8 does not recover the Shapley values. As previously mentioned, one consequence is that LIME violates Property 3, which can lead to unintuitive behaviour in certain circumstances.

Below we show how to avoid heuristically choosing the kernel in Equation 3.8. With a very specific weight, we can ensure that the solution of the Kernel SHAP linear weighted regression is such that the coefficients it returns are the Shapley values.

Theorem 3.4 (Shapley Kernel). *Under the definition of additive feature attribution methods, the specific forms of $\pi_{x'}$, $\mathcal{L}$, Ω that make solutions of Equation 3.8 consistent with Properties 1,2, and 3 are:*

$$\begin{aligned}\Omega(g) &= 0,\\ \pi_{x'}(z') &= \frac{M-1}{\binom{M}{|z'|}|z'|(M-|z'|)},\\ \mathcal{L}(f,g,\pi_{x'}) &= \sum_{z'\in\mathcal{Z}}[f(h_x^{-1}(z')) - g(z')]^2\,\pi_{x'}(z'),\end{aligned}$$

where $|z'|$ is the number of non-zero elements in z', and $\mathcal{Z}$ is the training data.

Proof. See supplementary material of (Lundberg and Lee, 2017)[39] and (Pei, 2018)[47]. □

In Kernel SHAP step 2 we sample K instances creating a background dataset B, but if these instances are not representative we put too much weight on unlikely data points. We can be a little more selective in our coalition sampling as the smallest and largest coalitions have the most of the weight. Instead of sampling blindly, we use some of the sample budget K to include these high-weight coalitions, resulting in better Shapley value estimations. We begin with all feasible coalitions with 1 and $M-1$ features, resulting in a total of 2 times M coalitions. We may incorporate coalitions with 2 features, $M-2$ features, and so on when we have enough budget left. We sample with readjusted weights from the remaining coalition sizes.

Formalising concepts from step 3, we construct a linear regression model

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

which is optimised by the loss function $\mathcal{L}$ presented in Equation 3.10. As proved in Theorem 3.3 the Shapley values are the estimated coefficients. As we are in a linear regression situation, we could think in doing the model sparse, including regularisation terms. However, the estimated coefficients would not be the Shapley values.

Although it is not discussed in this project, there are other ways SHAP is presented in the paper, but these are not model agnostic. They are rather model specific and aim to optimise the sampling process.

3.9.4 SHAP limitations

- **Computational cost**: Kernel SHAP is slow. As a result, while it can explain a single instance in a fair amount of time, it is unable to do so for a huge number of them. As a result, Kernel SHAP is unsuitable for explaining models globally, as such explanations are typically derived from explaining a large number of representative instances. One example of this computationally expensive global methods is *SHAP feature importance* (Molnar, 2022, Ch. 9.6.5)[21].

- **Feature dependencies**: Kernel SHAP does not take into account feature dependencies. By replacing feature values with values from random instances, it is easier to randomly sample from the *marginal distribution*[8] rather than taking into account the dependencies between the missing and non-missing features. This is fine as long as the features are independent. However, if features are correlated this might lead to sampling feature values that do not make sense for our instance. One solution might be conditional sampling; features are sampled based on the features that are already in the coalition. While conditional sampling solves the problem of unrealistic data points, it also introduces a new problem: the generated values are no longer Shapley values[9].
- **Access to data**: Contrary to LIME, calculating the Shapley value for a new data instance requires access to the data. Accessing the prediction function is not enough, because we are required to replace sections of the instance of interest with values from other randomly chosen instances of the data.

3.10 LIME VS SHAP

The information of this section has been extracted from (Molnar, 2022)[21] complemented with (Sundararajan, 2020)[48] and (Poduska, 2018)[49].

Remember that LIME builds a new dataset of perturbed data points. As these points are obtained from perturbing the instance of interest, in their majority, they fall near the instance. Then LIME trains a local surrogate model on it, with each sampled instance weighted by its proximity to the instance of interest. Kernel SHAP is similar to LIME in that it trains a local surrogate model, but the distinction is that it weights the sampled instances according to the weight the coalition would receive in a Shapley value estimation. For training the surrogate model, SHAP either can use the whole training set or a background dataset, which is formed by a subset of representative instances of the training set.

Shapley values take into account all possible predictions for an instance based on all possible input combinations. Due to this exhaustive approach, SHAP is the only technique that can guarantee local accuracy, missingness, and consistency, whereas LIME, as a particular case of SHAP, lacks the same properties.

So, why would anyone use LIME in the first place? It is simple, LIME is faster, and Shapley values require a considerable time to compute.

3.11 LIME and SHAP medical sector application

In this section, we present simulated user experiments to evaluate the utility of local explanations from LIME and SHAP in trust-related tasks. We show how using these

[8] Marginal distribution: Given two random variables together X and Y, the marginal distribution of X is simply the probability distribution of X ignoring the information regarding the variable Y.

[9] See (Sundararajan, 2020)[48].

two XAI methods, one can understand the influence of the variables on the ML model's decisions for particular instances, without having to peek into the model's internal parameters.

In order to emulate a real-life context in the field of healthcare, suppose we are in the position of a doctor who is seeking support to make a decision based on the prediction of a black box model. More specifically, in the position of a doctor who has to decide whether to diagnose a patient as diabetic or not. Our black box model will be a random forest, introduced in Chapter 2 and on which we have described two global explanations for the model through the importance of features, Gini importance and permutation importance. We will see that with LIME and SHAP this global importance of features does not always correspond to the features that have most influenced specific predictions.

The database used to train this model is a tabular database with diabetes health indicators, therefore, where the problem established is to diagnose each patient as either diabetic (or prediabetic) or not.

Python has been the programming language used for the modelling of this section and the elaboration of all the figures. The commented code and data for replicating the experiments are available at <https://github.com/aleixnieto/TFG>.

3.11.1 Diabetes database

The database on which we will work this section is a modified database coming from the *Behavioural Risk Factor Surveillance System (BRFSS)*, which is a health-related telephone survey collected annually by the *Centers for Disease Control and Prevention (CDC)* in the United States. Every year, BRFSS collects responses from over 400,000 Americans on health-related risk behaviours, chronic health conditions, and the use of preventative services. The used database[10] is a modified version of the 2015 BRFSS original database made available on *Kaggle*[11]. The original database contained responses from 441,455 individuals and has 330 features, and it has been cleaned and balanced to obtain a dataset of 70,692 individuals with 21 interpretable features. The balancing has been done by the database creator by selecting random instances until reaching an equal 50/50 split of respondents with no diabetes and with either prediabetes or diabetes.

The target variable in the database is defined as:

$$Y = "Diabetes" = \begin{cases} 0 & if \quad \text{Respondent with no diabetes or prediabetes} \\ 1 & if \quad \text{Respondent with either diabetes or prediabetes} \end{cases}$$

A description of the 21 predictor features can be found in *Appendix H*. Also, an exploratory data analysis of the database can be found at <https://github.com/aleixnieto/TFG/blob/main/PROFILING/Database_general_report.html>.

[10] https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset?select=diabetes_binary_5050split_health_indicators_BRFSS2015.csv.

[11] Kaggle: Online community platform for data scientists and machine learning enthusiasts. Kaggle allows users to collaborate with other users, find and publish datasets, and compete with other data scientists to solve data science challenges.

3.11.2 Random forest implementation

The goal of this subsection is to estimate and evaluate a machine learning predictive model based on our diabetes database. We are not intended to produce the most accurate predictive model, but a model on which to apply the previously discussed XAI techniques. To do this, we will use a random forest, which we introduced in Chapter 2 and of which we already have a solid understanding of the theory behind it.[12]

Before training the model, we first need to split the dataset into train and test datasets. Seventy-five per cent (75%) of the total data comprises the train dataset, the remaining twenty-five per cent (25%) forms the test dataset, which will be stored to later evaluate the model performance and generalisation ability on new unseen instances.

The random forest model generates independent trees using a subset of all available features and bootstrap observations from the train dataset, producing a unique structure for each tree. Therefore, the nature of random forest requires hyperparameters to generate each individual decision tree rather than having a fixed structure. The hyperparameters yielding the best performance of the model are selected through an hyperparameter tuning procedure.

The function *RandomizedSearchCV* from the Python library *sklearn* has been used to perform hyperparameter tuning. *RandomizedSearchCV* selects a fixed number of parameter settings at random from all possible parameter combinations of Table 3.1. By sampling 20 parameter settings at random from the parameter grid, the selection of the best parameters has been achieved via a 5-fold cross-validation.

The parameters proposed for the hyperparameter tuning are the following:

Hyperparameter	Description	Proposed values
n_estimators	Number of trees in the random forest.	{50,100,150,200,250}
max_features	Denote M the total number of model available features. The hyperparameter max_features $\leq M$ is the number of features considered at each split.	$\{\sqrt{M}, M\}$
min_samples_split	Minimum number of samples required to split an internal node.	{2, 4, 6, 8, 10}
min_samples_leaf	Minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least "min_samples_leaf" training samples in each of the left and right branches.	{1, 2, 3, 4}

Table 3.1: Hyperparameters used to train the random forest classifier with the corresponding hyperparameters description and its proposed values.

[12] Several comments looking for detail in the explanation are omitted in the following discussion. However, they can be found in the code.

After running the *RandomizedSearchCV* with a random forest classifier and the grid from Table 3.1 grid as inputs, the following hyperparameters have been obtained:

n_estimators	max_features	min_samples_split	min_samples_leaf
250	$\sqrt{M}$	2	4

Table 3.2: Selected hyperparameters in the random forest classifier. The selection has been performed via a 5-fold cross-validation using *accuracy* as evaluation metric.

Once obtained the hyperparameters, we can evaluate the random forest performance by looking at different validation metrics in our training and test datasets:

Metric	Training set	Test set
Accuracy	0.834	0.753
Precision	0.812	0.730
Recall	0.871	0.799
F-score	0.840	0.763

Table 3.3: Comparison of 4 performance metrics between train and test datasets. The better performance in the training set is due to these data being observed during training, in contrast to test instances that remain unseen. Such little difference ensures low overfitting and, hence, the good generalisation of the model. See (Chauhan, 2020)[50] for a detailed insight into these metrics.

After evaluating and training our random forest model, we can obtain a global explanation through the importance of the variables. This is achieved by the Gini importance and permutation feature importances. In Figure 10 we compare them with our diabetes database features.

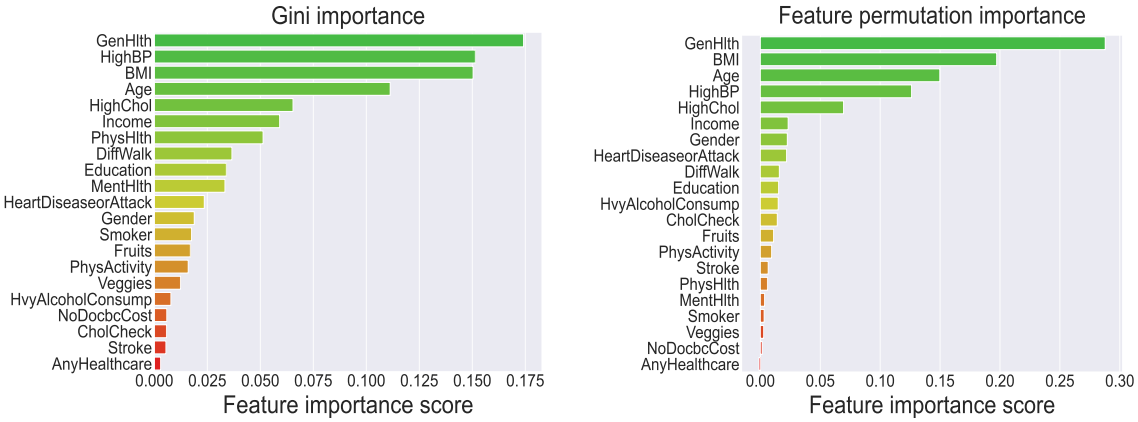


FIGURE 10: Global importances based on Gini and feature permutation importances. The features are ordered from highest to lowest according to their contribution. As the feature permutation importance gives us the change in the model's performance, its values have been standardised to be represented on a scale relative to 1.

We can observe that in both cases the 5 most important variables coincide. However, in the case of feature permutation importance, the variation in model performance is significantly greater in these variables than in the rest.

3.11.3 Visualising a single instance explanation

This section shows an example where LIME and SHAP help us to build confidence in our random forest, which plays the role of an arbitrary black box model. To achieve this, we examine the explanation of an incorrectly predicted instance from the test dataset. We locally look at which variables have led to the prediction and examine them as a reason to validate the fact that they make sense from a medical point of view. Therefore, translating this concept into a real medical environment, if a doctor saw in the explanation of a prediction that the variables that led to the prediction do not make sense in medical terms, he would not accept the prediction as valid. Also, emphasising that the chosen database is just used as an example since its features are well-known and do not require in-depth medical expertise to interpret and analyse.

LIME

To obtain the LIME's explanation for an individual instance, the LIME author's Python implementation[13] has been used. In Figure 11 we can see the LIME explanation for an instance which has been incorrectly predicted by the random forest. Concretely, the individual has been predicted as diabetic when in fact the person was not diabetic.

Feature	Value	Weight
GenHlth	2	-0.191825
HighBP	0	-0.151779
HighChol	1	0.109313
HeartDiseaseorAttack	0	-0.081861
HvyAlcoholConsump	0	0.080394
Age	11	0.074182
BMI	33	0.060582
DiffWalk	1	0.052797
Income	4	0.046733
Stroke	0	-0.029359
Gender	0	-0.023914
Education	4	0.023293
PhysHlth	0	-0.007371
PhysActivity	0	0.005999
Fruits	1	-0.005114
MentHlth	1	-0.004775
NoDocbcCost	0	0.003120
Smoker	1	-0.000084
CholCheck	1	0.000000
Veggies	1	0.000000
AnyHealthcare	1	0.000000

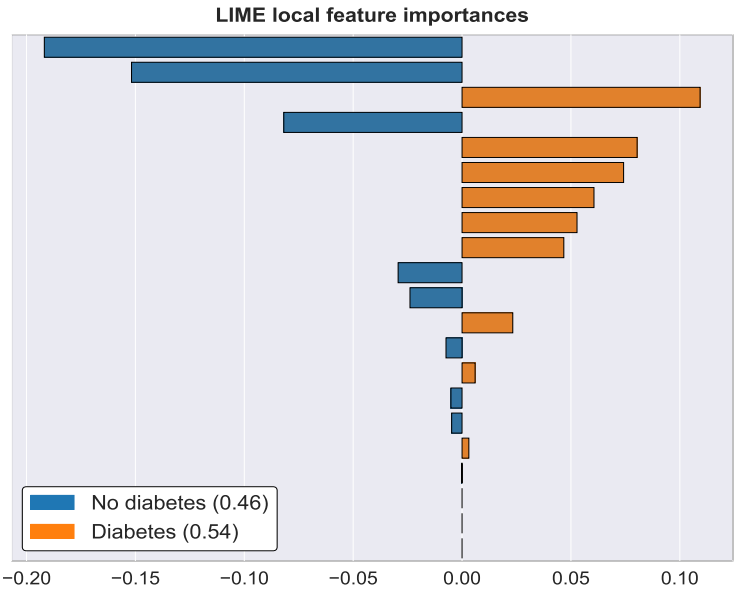


FIGURE 11: LIME explanation for a single instance. In the table we can see the values that take the features of the instance being explained, which are ordered according to their weight in absolute value. The blue colour represents that the corresponding feature has contributed to the prediction being "No Diabetes", and the orange colour to the prediction being "Diabetes". To train the surrogate model from which we received the weights, 20,000 perturbed instances around the predicted instance were generated.

In Figure 11 we can see that the majority voted class was "Diabetes" by 54% of the trees. But is this really faithful? Note that the second variable that most contributed to the prediction of diabetes is "*HvyAlcoholConsump*", but this does not make sense as "*HvyAlcoholConsump*" takes the value 0, which corresponds to a person who does not consume alcohol regularly. In this case, again positioning ourselves in the position of a doctor, the explanation would lead us to reject the model's prediction due to a lack of coherence in the reasons that have led to said prediction.

[13] https://github.com/marcotcr/lime.

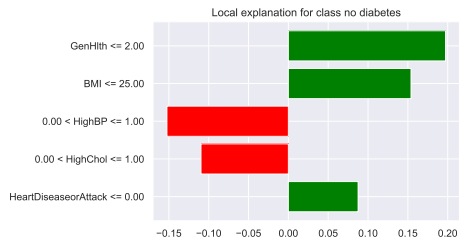
Additionally, after analysing many individual explanations one by one, we can conclude that the model has interpreted "*HvyAlcoholConsump*" in the opposite direction. It contributes to diabetes when it takes the value 0, which is a person who does not consume alcohol, and it contributes to non-diabetes when it takes the value 1, which is a person who consumes alcohol. This reasoning of the model does not make sense and therefore we have discovered a feature that globally affects the model's output. Note that we have achieved a global issue from the model by analysing individual instances.

After running the explanation many times we can conclude that the perturbed instances generated around the explained instance are stable with the LIME's output as the explanation has always been the same. Also, note that it has been used the default kernel width implemented by the LIME's author and already commented in *Appendix F*.

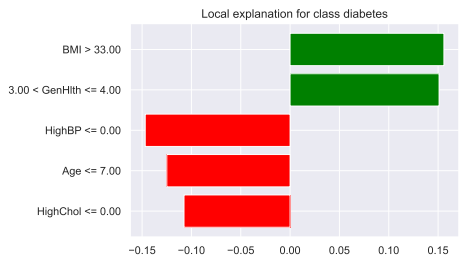
As a general conclusion, after inspecting many instances, it can be said that except for the feature "*HvyAlcoholConsump*", all the others have been interpreted by the model as it would be expected following a medical sense[14]. Moreover, the features that are usually the most important in the predictions are also the 5 most important in Figure 10.

SP-LIME

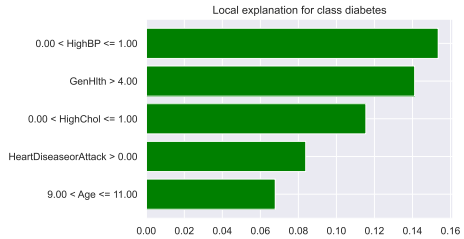
In this part we run the SP-LIME algorithm to our database just as an example. The figures have been generated by the LIME'S author Python implementation. In our case, SP-LIME selects the 4 instances with the highest information coverage. For the explanations we have selected 5 as the number of features we want to explain the prediction, that is, the 5 most important features that lead to the prediction.



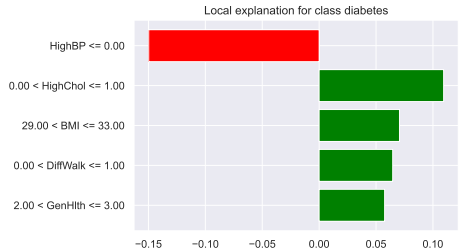
(a) First selected instance



(b) Second selected instance



(c) Third selected instance



(d) Fourth selected instance

FIGURE 12: SP-LIME algorithm calculated from 500 candidate explanations sampled from the data uniformly at random. Without this sampling, explanations would be generated for the entire dataset, which is time-expensive. By analysing these explanations we can get a general grasp of how the algorithm works. Red contributes to non-diabetes and green contributes to diabetes.

[14] Based on the diabetes risk factors from https://www.cdc.gov/diabetes/basics/risk-factors.html

In Figure 12, looking at the features' values and their contributions we can see that all the explanations are reasonable except for the one from Figure 12a. The first selected instance has been predicted as non-diabetic and the variables that have contributed the most are "*HighBP*" and "*HighChol*" taking the value 0. This does not make sense in medical terms as having high blood pressure and having a heart attack are clear risk factors for developing diabetes or prediabetes. Then again we have an explanation that would lead us to reject the prediction.

SHAP

To obtain the SHAP plots, the SHAP author's Python implementation[15] has been used.

In Figure 13 we can see the SHAP explanation for the same prediction explanation in LIME's example. The output value 0.54 has been marked in bold and it is the prediction probability for the instance being predicted as diabetic. That is, 54% of the decision trees in the forest voted for the diabetes class, so the instance was predicted as diabetic.

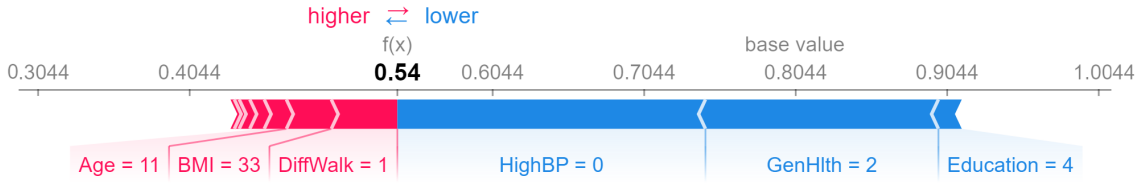


FIGURE 13: Explanation for a single instance obtained with the function KernelExplainer from the Python's SHAP library. Red features contribute to diabetes prediction and are represented by pushing the prediction higher, and those pushing the prediction lower are in blue, and contribute to a non-diabetic prediction.

The *base value*, according to the original SHAP paper, is the mean of the model output over the background dataset. In our case to impute the missing values for the features excluded in the Kernel SHAP algorithm, we have used the median of the train dataset.

Therefore, our background dataset is formed by one instance where each feature takes its median in the train dataset. Taking the prediction for this instance we get 0.8044, which is the probability that the instance is predicted to be diabetic, which is exactly the base value.

In Figure 13 we see how non-high blood pressure, good general health contribute in this order of importance to a non-diabetic prediction. Contrarily, the difficulty of walking, high body mass index and an advanced age have a positive impact on the diabetes prediction. In this case the explanation makes sense and consequently it is a good complement to the model prediction.

SHAP framework also offers a barplot to observe, in a more intuitive way, the feature contributions *(See Appendix I)*.

Although it is not covered in this project, SHAP develops a global explanation theory from the local theory we have already seen. This SHAP's global framework, is by far, preferable to SP-LIME. See (Molnar, 2022, Ch. 9.6)[21] and (Kuo, 2019)[51] for a detailed insight into SHAP global explanation theory approach.

[15] https://github.com/slundberg/shap.

Conclusion

We have formalised the theory behind LIME and SHAP. Moreover, we have seen how, for the same instance, the explanations given by these two techniques are not equal. This discrepancy in the explanation is due to a difference in the very nature of the models. LIME tells us, in a local sense, which are the most important features around the instance of interest and its biggest drawback is that the explanation varies depending on the kernel width. Otherwise, SHAP is consistent with describing how Shapley values decompose the final prediction into the contributions of each feature. But as LIME with the kernel width, we have some control over a parameter, in this case the decision about how the omitted features are imputed.

By analysing these two methods in detail we have seen which are their strengths and weaknesses. The table below summarises how LIME and SHAP differ on certain characteristics that define the application context of the two methods:

Property	**LIME**	**SHAP**
Theory driven	Fails at being consistent. ✗	Supported by the Shapley values theory properties and consistency property. ✓
Time expensive	Time affordable. ✓	Computation of marginal contributions for all possible coalitions makes it time expensive. ✗
Require training data	Does not require the training set for fitting the surrogate model. cmark	Requires the training set for generating the background set that will be used to train the surrogate model. ✗
What-if-explanations	Can provide what-if explanations.[16]	Cannot provide what-if explanations. ✗
Improbable instances	Improbable instances may be generated when obtaining perturbed instances. ✗	When imputing omitted features, improbable instances may be generated. ✗
Instability	Kernel width can make it unstable. ✗	Its strong theoretical properties makes it stable. ✓

Table 3.4: Comparative table of LIME and SHAP based on different criteria.

After having seen an application of these techniques in the field of healthcare, we may wonder which of these two methods would be appropriate to use depending on the context. Based on the strong theoretical properties of SHAP, it would be preferable to use SHAP when we can, and rely on LIME when SHAP's compute costs are too high.

[16] LIME can provide what-if explanations by plugging in different values of features into the local model and measuring how the prediction changes.

Future work

There are a number of avenues of future work that could be further explored. The first thing which could be investigated in the future is to see how LIME explanations vary depending on the kernel width. The idea would be to see for which kernel width value the explanations are stable. This would be achieved by observing how the weights of the explanation behave, and if this depends on the selected instance. The second way in which we would like to complement the project is to expand LIME and SHAP application to images in the field of health. We have seen how the explanations of LIME and SHAP are interpreted. However, these have not been compared from a numerical point of view. Therefore, another possible extension would be to see through some kind of metric how the LIME and SHAP explanations relate.

References

[1] D. **Silver**, A. **Huang**, C. J. **Maddison**, A. **Guez**, L. **Sifre**, G. **Van Den Driessche**, J. **Schrittwieser**, I. **Antonoglou**, V. **Panneershelvam**, M. **Lanctot**, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, **2016**.

[2] J. **Sarzynska-Wawer**, A. **Wawer**, A. **Pawlak**, J. **Szymanowska**, I. **Stefaniak**, M. **Jarkiewicz**, and L. **Okruszek**, "Detecting formal thought disorder by deep contextualized word representations," *Psychiatry Research*, vol. 304, p. 114135, **2021**.

[3] S. M. **Lundberg**, B. **Nair**, M. S. **Vavilala**, M. **Horibe**, M. J. **Eisses**, T. **Adams**, D. E. **Liston**, D. K. **Low**, S. **Newman**, J. **Kim**, *et al.*, "Explainable machine-learning predictions for the prevention of hypoxaemia during surgery," *Nature biomedical engineering*, vol. 2, no. 10, pp. 749–760, **2018**.

[4] M. **Mohri**, A. **Rostamizadeh**, and A. **Talwalkar**, *Foundations of machine learning*. MIT press, **2018**.

[5] A. L. **Samuel**, "Some Studies in Machine Learning Using the Game of Checkers," *IBM Journal of Research and Development*, vol. 3, pp. 210–229, Jul **1959**.

[6] D. **Fumo**, "Types of machine learning algorithms you should know," Aug **2017**. https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-953a08248861.

[7] S. **Fortmann-Roe**, "Bias and variance," Jun 2012. http://scott.fortmann-roe.com/docs/BiasVariance.html.

[8] M. **Bourel**, "Model aggregation methods and applications," *Memoria Investigaciones en Ingeniería*, **2012**.

[9] T. **Hastie**, R. **Tibshirani**, and J. H. **Friedman**, *The elements of statistical learning: data mining, inference, and prediction*, vol. 2. Springer, **2009**.

[10] J. **Brownlee**, "Classification and regression trees for machine learning," Aug **2020**. https://machinelearningmastery.com/classification-and-regression-trees-for-machine-learning/.

[11] D. **López**, "The complete guide to decision trees," Apr **2019**. https://towardsdatascience.com/the-complete-guide-to-decision-trees-28a4e3c7be14.

[12] B. **Li**, J. **Friedman**, R. **Olshen**, and C. **Stone**, "Classification and regression trees," *Biometrics*, vol. 40, no. 3, pp. 358–361, **1984**.

[13] B. D. **Ripley**, "Neural networks and pattern recognition," *Cambridge University*, vol. 7, **1996**.

[14] J. **Rocca**, "Ensemble methods: Bagging, boosting and stacking," Apr 2019. https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205.

[15] J. **Frost**, "Introduction to bootstrapping in statistics with an example," **2018**. https://statisticsbyjim.com/hypothesis-testing/bootstrapping/.

[16] J. **Llano**, "Modelització dels risc de crèdit a través del machine learning," June **2020**.
http://diposit.ub.edu/dspace/handle/2445/177698?mode=full.

[17] L. **Breiman**, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, **1996**.

[18] T. G. **Dietterich**, "Ensemble methods in machine learning," in *International workshop on multiple classifier systems*, pp. 1–15, Springer, **2000**.

[19] J. **Surowiecki**, "The wisdom of crowds: Why the many are smarter than the few and how collective wisdom shapes business," *Economies, Societies and Nations*, vol. 296, no. 5, **2004**.

[20] L. **Breiman**, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, **2001**.

[21] C. **Molnar**, "Interpretable machine learning," **2022**.
https://christophm.github.io/interpretable-ml-book/.

[22] T. **Parr**, "Beware default random forest importances," Mar **2018**.
https://explained.ai/rf-importance/index.html.

[23] L. **Simon** and D. **Young**, "Lesson 1: Simple linear regression," **2022**.
https://online.stat.psu.edu/stat501/lesson/1.

[24] L. **Simon** and D. **Young**, "Lesson 13: Weighted least squares & robust regression," **2022**.
https://online.stat.psu.edu/stat501/lesson/13.

[25] R. **Vaghefi**, "Weighted linear regression," Feb **2021**.
https://towardsdatascience.com/weighted-linear-regression-2ef23b12a6d7.

[26] E. **Wisam**, *Weighted Least Squares: Solution Derivation*. YouTube, Oct **2020**.
https://www.youtube.com/watch?v=_yn3F8h4Nkc&ab_channel=EssamWisam.

[27] P. **Gupta**, "Regularization in machine learning," Nov **2017**.
https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a.

[28] V. **Chekka**, "Regularization in machine learning: Connect the dots," Sep **2018**.
https://towardsdatascience.com/regularization-in-machine-learning-connecting-the-dots.

[29] H. **Phong**, "Regulaziation: Ridge and lasso," Jan **2022**.
https://medium.com/@IwriteDSblog/the-regularized-models-9ed187c0bfc9.

[30] A. E. **Hoerl** and R. W. **Kennard**, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, **1970**.

[31] P. P. **Angelov**, E. A. **Soares**, R. **Jiang**, N. I. **Arnold**, and P. M. **Atkinson**, "Explainable artificial intelligence: an analytical review," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 11, no. 5, p. e1424, **2021**.

[32] E. **Tjoa** and C. **Guan**, "A survey on explainable artificial intelligence (xai): Toward medical xai," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 11, pp. 4793–4813, **2020**.

[33] M. T. **Ribeiro**, S. **Singh**, and C. **Guestrin**, ""why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144, **2016**.

[34] M. T. **Ribeiro**, "Lime - local interpretable model-agnostic explanations," **2016**. https://homes.cs.washington.edu/ marcotcr/blog/lime/.

[35] **DeepFindr**, *Explainable AI explained! | #3 LIME*. YouTube, Feb **2021**. https://www.youtube.com/watch?v=d6j6bofhj2M&t=478s&ab_channel=DeepFindr.

[36] A. **Sharma**, "Decrypting your machine learning model using lime," Nov **2018**. https://towardsdatascience.com/decrypting-your-machine-learning-model-using-lime-5adc035109b5.

[37] D. **Alvarez-Melis** and T. S. **Jaakkola**, "On the robustness of interpretability methods," *arXiv preprint arXiv:1806.08049*, **2018**.

[38] G. **Knor**, *LIME for NLP - Explainable AI*. YouTube, Aug **2021**. https://youtu.be/a4D1_Gpgz_c?t=1581.

[39] S. M. **Lundberg** and S.-I. **Lee**, "A unified approach to interpreting model predictions," *Advances in neural information processing systems*, vol. 30, **2017**.

[40] M. **Sahakyan**, Z. **Aung**, and T. **Rahwan**, "Explainable artificial intelligence for tabular data: A survey," *IEEE Access*, vol. 9, pp. 135392–135422, **2021**.

[41] **Machine Learning TV**, *Understanding The Shapley Value*. YouTube, Oct **2021**. https://www.youtube.com/watch?v=9OFMRiAVH-w&ab_channel=MachineLearningTV.

[42] **DeepFindr**, *Explainable AI explained! | #4 SHAP*. YouTube, Mar **2021**. https://www.youtube.com/watch?v=9haIOplEIGM&t=490s&ab_channel=DeepFindr.

[43] **Kie**, *Shapley Additive Explanations (SHAP)*. YouTube, Jun **2021**. https://www.youtube.com/watch?v=VB9uV-x0gtg&t=219s&ab_channel=KIE.

[44] L. S. **Shapley**, "Notes on the n-person game-ii: The value of an n-person game," **1951**.

[45] C. **Molnar**, "characteristics of shapley values." Cross Validated, **2019**. https://stats.stackexchange.com/q/389261.

[46] L. **Shapley**, "A value for n-person games, in contributions to the theory of games, ed. by hw kuhn," *Princeton University Press*, pp. 307–317, **1953**.

[47] Y. **Pei**, "Shapley, lime and shap," Dec **2018**. https://ypei.org/posts/2018-12-02-lime-shapley.html.

[48] M. **Sundararajan** and A. **Najmi**, "The many shapley values for model explanation," in *International conference on machine learning*, pp. 9269–9278, PMLR, **2020**.

[49] J. **Poduska**, "Shap and lime: Great ml explainers with pros and cons to both," Dec **2018**.
https://blog.dominodatalab.com/shap-lime-python-libraries-part-1-great-explainers-pros-cons.

[50] N. S. **Chauhan**, "Model evaluation metrics in machine learning," May **2020**.

[51] C. **Kuo**, "Explain your model with the shap values | by chris kuo/dr. dataman | towards data science," Sep **2019**.

[52] M. **Pham**, "Regularization and geometry," May **2019**.
https://towardsdatascience.com/regularization-and-geometry-c69a2365de19.

[53] G. **Visani**, "Lime: Explain machine learning predictions," Dec **2020**.
https://towardsdatascience.com/lime-explain-machine-learning-predictions-af8f18189bfe.

[54] G. **Visani**, E. **Bagli**, and F. **Chesani**, "Optilime: Optimized lime explanations for diagnostic computer algorithms," *arXiv preprint arXiv:2006.05714*, **2020**.

Appendices

Appendix A

In order to have simple and visual representation, we will use two variables, X and Y. In the following bi-dimensional example, the decision tree partitions the training data into a set of rectangles.

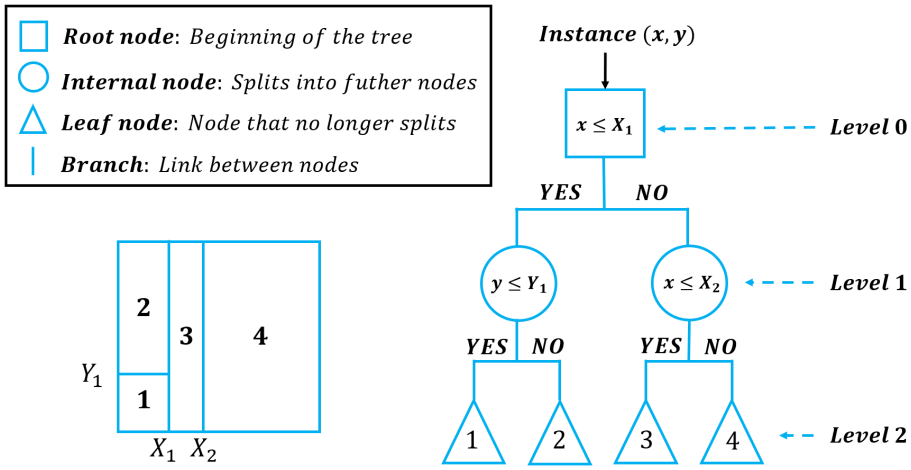


FIGURE 14: Decision tree of two levels. Picture inspired by (López, 2019)[11].

Each node represents a feature, each branch represents a rule (or decision), and each leaf represents an outcome. The depth of a tree is defined by the number of levels, not including the root node. Given the training data, the decision tree tries to group observations that are similar among them, and look for the best rules that split the observations that are dissimilar between them until they reach a certain degree of similarity.

Appendix B

The three metrics of node impurity are comparable in Figure 16; however, cross-entropy and Gini index are differentiable and hence more accessible to numerical optimisation.

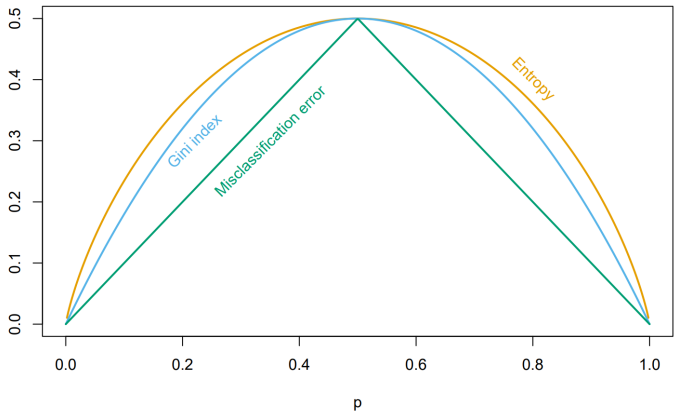


FIGURE 15: Node impurity measures for binary classification, as a function of the proportion p in the second class. Cross-entropy has been scaled to pass through $(0.5, 0.5)$. Image adopted from (Hastie, 2009)[9].

Appendix C

In this figure, we can see how increasing the number of weak learners, each with a chance of correctly producing the right decision larger than 0.5, decreases the probability of wrongly predicting the class of an instance. This occurs when the total number of models that predict class 1 is less or equal than half of the total number of models.

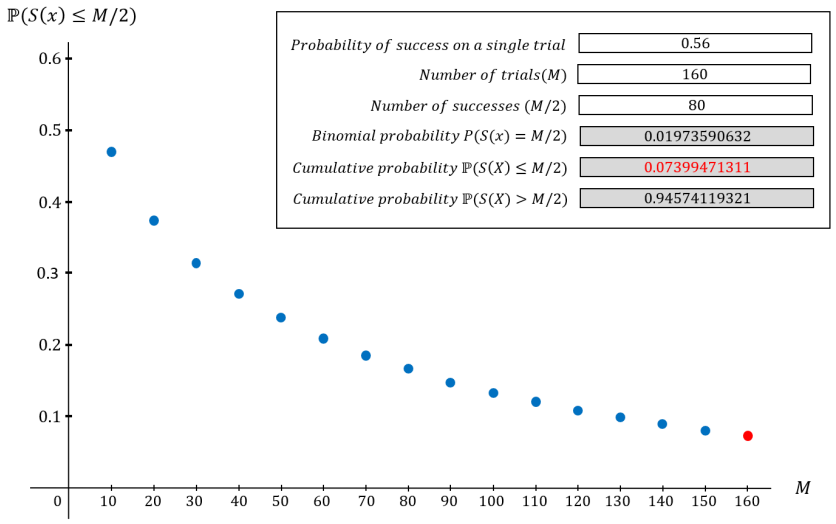


FIGURE 16: Here, our weak learners are represented as trials in a binomial experiment. For this example we have used p=0.56 as the probability of success of each trial. The probability that, given an instance and 160 weak learners, we incorrectly classify this instance half or more of the time is indicated in red.

Appendix D

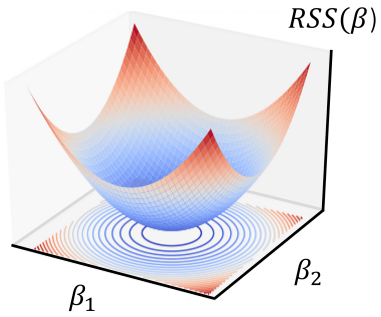


FIGURE 17: Gradient descent projection as contour, adapted from (Chekka, 2019)[28].

In Figure 18, gradient descent is seen in three dimensions. The error for corresponding β_1 and β_2 is shown by $RSS(\beta)$. The zones with the highest error are shown in red, while the zones with the smallest error are represented in blue. The coefficients β_1 and β_2 would be determined using gradient descent where the error is the global minimum, resulting in $\hat{\beta}$. The relevant error from the $RSS(\beta)$ cost function of gradient descent is projected on the 2-dim with the same colours.

Appendix E

In a 2-D space, the set of betas that we can "afford" with L1 regularisation lies within a rhombus, which we have called a "diamond". If the red point illustrated in Figure 4 is at the corner of the diamond, one of the betas is set to 0. Contrarily, if the ellipse reaches the diamond on the edge, none of the betas becomes 0, being this a rare situation.

Now we consider the situation when we have three betas (3-D space). The constraint geometry is as follows:

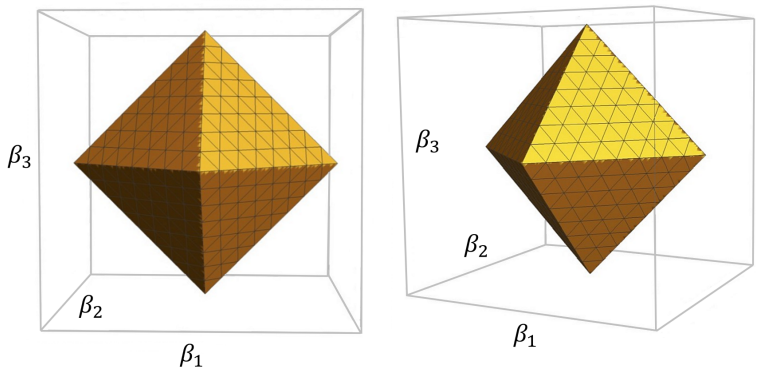


FIGURE 18: Lasso with 3 betas, adapted from (Pham, 2019)[52].

Figure 19 shows us that the constraint region becomes an octahedron; a shape with 8 faces, 12 edges and 6 vertices. As the dimension increases, the number of vertices and edges increases as well, making the ellipse more likely to be in contact with the "diamond" on one of those places. That being said, lasso tends to work better in higher dimensions.

Appendix F

The information of this appendix has been extracted from (Visani, 2020)[53].

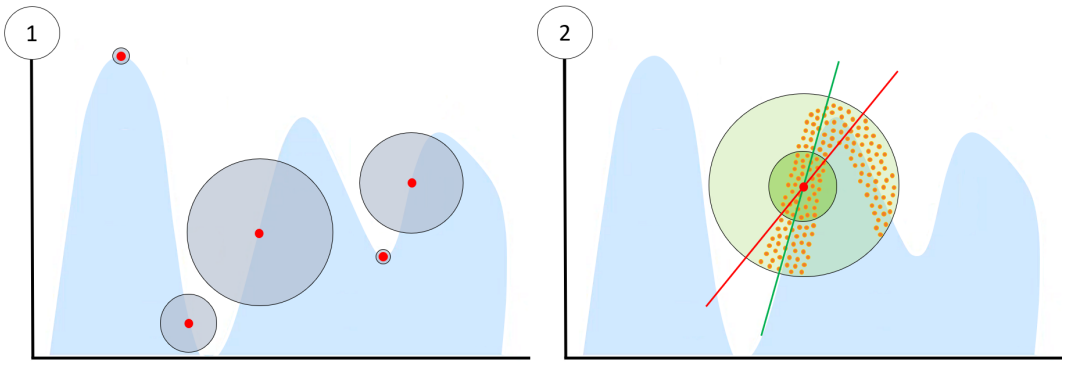


FIGURE 19: (1) The best neighbourhood size depends on the target point (red points) and the curvature of the ML function around it. **(2)** In a very intuitive way, orange dots are the original dataset points used to train our original complex model. Green circles show how the kernel weights are assigned, based on the kernel width parameter. The inner circle gives meaningful weights only to very close instances because σ^2 is small. The outer circle employs a larger σ^2.

The diameter of the neighbourhood is determined by the kernel width: a small kernel width indicates that an instance must be quite close to impact the local model, whereas a greater kernel width indicates that further away instances influence the model as well.

LIME author's Python implementation uses a kernel width of 0.75 times the square root of the number of columns in the training data. However, we argue this is very dependent on the data. In Figure 20 (left) we can see how different points have different proper size for the linear local region. Moreover, in Figure 20 (right) we can see how bad choices of the kernel width distort the LIME surrogate model approximation, making it very different from the local shape of our original model. The green line only takes into points in a small region, whereas the red line is the surrogate linear model trained in a major diameter where the nonlinear form of the complex model affects its direction.

Recently, a new framework called *OptiLIME* presented in (Visani, 2020)[54] has been developed to find the best kernel width so that LIME explanation is ensured to represent the tangent to the ML curve.

Appendix G

Possible joining orders *Ways of adding player 1 to form the grand coalition*	**Marginal contributions of player 1** $v(S \cup \{1\}) - v(S)$	**Assigned weight to marginal contribution** $\frac{\lvert S\rvert!(\lvert N\rvert-\lvert S\rvert-1)!}{\lvert N\rvert!}$
$\{1\} \to \{1,2\} \to \{1,2,3\}$	$v(\{1\}) - v(\{\emptyset\})$	1/3
$\{1\} \to \{1,3\} \to \{1,3,2\}$	$v(\{1\}) - v(\{\emptyset\})$	1/3
$\{2\} \to \{2,1\} \to \{2,1,3\}$	$v(\{1,2\}) - v(\{2\})$	1/6
$\{2\} \to \{2,3\} \to \{2,3,1\}$	$v(\{1,2,3\}) - v(\{2,3\})$	1/6
$\{3\} \to \{3,1\} \to \{3,1,2\}$	$v(\{1,3\}) - v(\{3\})$	1/3
$\{3\} \to \{3,2\} \to \{3,2,1\}$	$v(\{1,2,3\}) - v(\{2,3\})$	1/3

Table 5: Three players example, that is $|N| = 3$. The different subsets S created by adding participants to build the grant coalition N are shown in the first column, concretely $3! = 6$. The marginal contributions of players in each of these adding possibilities are presented in the second column, and the weight assigned to each marginal contribution is presented in the third column.

Possible joining orders *Ways of adding players to form the grand coalition*	**Marginal contributions of player 1** $v(S \cup \{1\}) - v(S)$	**Marginal contributions of player 2** $v(S \cup \{2\}) - v(S)$	**Marginal contribution weight** $\frac{\lvert S\rvert!(\lvert N\rvert-\lvert S\rvert-1)!}{\lvert N\rvert!}$	**Shapley values** $(\phi_i(v))$ $\sum_{S \subseteq N \setminus \{i\}} \frac{\lvert S\rvert!(\lvert N\rvert-\lvert S\rvert-1)!}{\lvert N\rvert!}(v(S \cup \{i\}) - v(S))$	
				ϕ_1	ϕ_2
$\{1\} \to \{1,2\}$	$v(\{1\}) - v(\{\emptyset\}) = 1$	$v(\{2\}) - v(\{\emptyset\}) = 2$	1/2	1.5	2.5
$\{2\} \to \{1,2\}$	$v(\{1,2\}) - v(\{2\}) = 2$	$v(\{1,2\}) - v(\{1\}) = 3$	1/2		

Table 6: Two players example, that is $|N| = 2$. The different subsets S created by adding participants to build the grant coalition N are shown in the first column, concretely $2! = 2$. The marginal contributions of players in each of these adding possibilities are presented in columns two and three, and the weight assigned to each marginal contribution is presented in the fourth column. The Shapley values for player 1 and player two are displayed in the last two columns. The values used in this example are $v(\{1\}) = 1$, $v(\{2\}) = 2$, $v(\{1,2\}) = 4$.

Appendix H

Feature name	Descripted labels	Type
HighBP	0 = no high blood preasure; 1 = high blood preasure	Categorical
HighChol	0 = no high cholesterol; 1 = high cholesterol	Categorical
CholCheck	0 = no cholesterol check in 5 years; 1 = yes cholesterol check in 5 years	Categorical
BMI	Body Mass Index	Numerical
Smoker	Have you smoked at least 100 cigarettes in your entire life? [Note: 5 packs = 100 cigarettes] 0 = no; 1 = yes	Categorical
Stroke	Ever diagnosed with a stroke? 0 = no; 1 = yes	Categorical
HeartDiseaseorAttack	Coronary heart disease or myocardial infarction? 0 = no; 1 = yes	Categorical
PhysActivity	Realised physical activity in the past 30 days (not including job)? 0 = no; 1 = yes	Categorical
Fruits	Consume fruit 1 or more times per day? 0 = no; 1 = yes	Categorical
Veggies	Consume Vegetables 1 or more times per day? 0 = no; 1 = yes	Categorical
HvyAlcoholConsump	Adult men >= 14 drinks per week Adult women >= 7 drinks per week 0 = no; 1 = yes	Categorical
AnyHealthcare	Have any kind of health care coverage? 0 = no; 1 = yes	Categorical
NoDocbcCost	Was there a time in the past 12 months when you needed to see a doctor but could not because of cost? 0 = no; 1 = yes	Categorical
GenHlth	Would you say that in general your health is? scale 1-5 1 = excellent; 2 = very good; 3 = good; 4 = fair; 5 = poor	Categorical
MentHlth	Days of poor mental health in the past 30 days. Scale 1-30	Numerical
PhysHlth	Physical illness or injury days in the past 30 days. Scale 1-30	Numerical
DiffWalk	Do you have serious difficulty walking or climbing stairs? 0 = no; 1 = yes	Categorical
Gender	0 = female; 1 = male	Categorical
Age	13 - level age category (_AGEG5YR see codebook) 1 = 18-24; 9 = 60-64; 13 = 80 or older; etc.	Numerical
Education	6 - level education level (EDUCA see codebook) 1 = Never attended school; 2 = elementary; etc.	Categorical
Income	8 - level income scale (INCOME2 see codebook) 1 = less than $10,000; 8 = $75,000 or more; etc.	Categorical

Table 7: Description of the 21 predictor features of the diabetes database.

A much more detailed description of the features can be found at the BRFSS codebook <https://www.cdc.gov/brfss/annual_data/2015/pdf/codebook15_llcp.pdf>.

Appendix I

In Figure 21 we can see the explanation for the instance explained in Figure 14, now, in a more intuitive way using a barplot.

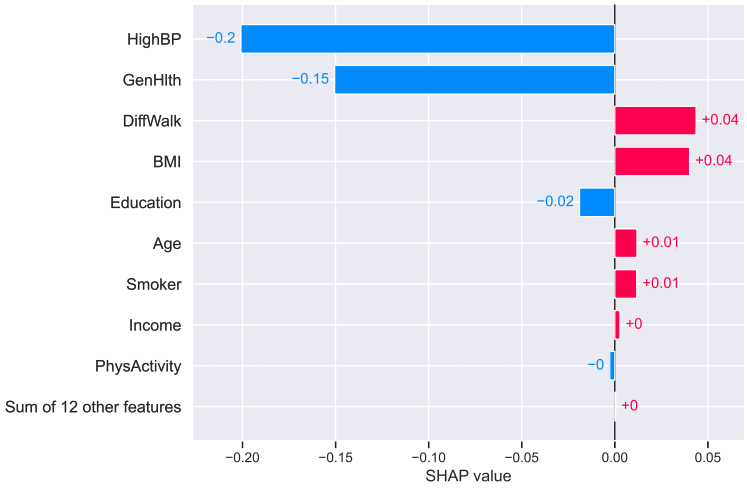


FIGURE 20: SHAP explanation for a single instance barplot.