Delft University of Technology

Master Thesis

Explanatory Techniques for Machine Learning Models

Author:
Klest Dedja

Supervisor:
Dr. Pasquale Cirillo

Committee:
Prof. C.W. Oosterlee
Dr. Den Iseger
Dr. F. Vermolen

6th November, 2019

If knowledge can create problems, it is not through ignorance that we can solve them.

Isaac Asimov

DELFT UNIVERSITY OF TECHNOLOGY

Abstract

Faculty of Applied Mathematics

Master Thesis

Explanatory Techniques for Machine Learning Models

by Klest Dedja

Since the last decade, we are assisting a widespread use of "black box" Machine Learning algorithms, these are algorithms with excellent performance but whose outcomes are hard to understand to a human agent. However, there are some situation when it is important to understand why a certain output is given, and the field of *explanability* in Machine Learning has flourished in the last decade.

In this work, we will go through some of these techniques. We will focus on model agnostic visualisation techniques introduced by Friedman (2001) and developed by Goldstein et al. (2015). Starting from the Partial Dependence plotting technique, we then analyse the Individual Conditional Expectation plot and its variants. Among them, we suggest the introduction of the so called "d-log-ICE" and we try identify scenarios where this techniques can bring better interpretability. We test our techniques on two models, the first one is based on the Boston Housing Dataset,the second is an internal model at ABN Amro.

Contents

Chapter 1

Introduction

Notes

This is the **public** version of the Thesis, some of the data from the host institution might be subject to non-disclosure agreements, we will therefore omit them in this version of the Thesis.

> "Machine Learning is a set of methods that computers use to make and improve predictions or behaviours based on data. For example, to predict the value of a house, the computer would learn patterns from past house sales following the steps of an algorithm."

This definition by Molnar (2019, pg. 10) is, among many others, a good definition of Machine Learning. In recent years this topic has been a lively field of research because of the applications which nowadays are possible, especially on the business side. Machine Learning is often considered as a sub-field of Artificial Intelligence (see for example Marr (2016)) as machines seem to behave in an intelligent way: they perform tasks which are considered "smart", and their predictive power comes from exploiting patterns in data rather than on a specified fixed algorithm. This kind of non-explicit programming is often called "indirect programming" and it takes place by providing data to the machine.

The field of Machine Learning (ML) has developed throughout the decades, the first mentions of it dating back to Samuel (1959), and it has been trending in the past few years thanks to the increased machines' computational power and to the enormous availability of data. Nowadays it is possible to run ML algorithms in a laptop for a personal project with very good results. This widespread availability of the technology has led many companies to base an increasing share of their decision-making processes based on ML algorithms and ABN Amro Bank, where I am performing my research work, is of course one of them. With such increased popularity ML-based models are being involved in more and more decision process, and a field which started by learning how to play

the games of checkers (Samuel, 1959) is now involved in all sorts of fields of knowledge: handwriting recognition, medical diagnosis, chess playing engines, YouTube's classification algorithm, search engines, or as in my case credit-risk modelling.

Such different applications lead to new needs and challenges for the modellers. In particular since the rise of highly performing "black box" models such as (Deep) Neural Networks, we are assisting a widespread use of algorithms with excellent performance but whose outcomes are hard to understand to a human agent. These new algorithms are thus enforcing a so called *interpretability vs. accuracy trade-off*, that is, they are enforcing a trend where the more performing the algorithm is, the higher the chances that their explainability is compromised, mainly due to the high complexity of interactions between the data and the output. Given such trend, finding a good balance between the two is the biggest challenge is this developmental stage of Artificial Intelligence, The question naturally rises: when are we willing to sacrifice clarity for performance in our tasks, delegating all responsibility to machines, and where should we try to reach a compromise with models whose outputs can be grasped by a human agent? The balance within the spectrum is found according to the context where the algorithm is used. In the next Section we will identify situations where explainability is highly desirable and where it is not. We will also end up identifying two factors that are taken into account to assess this.

1.1 Importance of explainability

The importance of explainability depends on the context and we will identify a couple of factors which influence where on the trade-off spectrum one is willing to land. We show four representative examples where explainability is little, somewhat desirable or very important for the scope of project.

In hand-writing recognition for example, there is no need for clarification as long as ML techniques have been successfully used for decades and there is historical evidence that they perform well. Moreover, a personal handwriting recognition algorithm does not involve big stakes and the price to pay in case of an error is small. Such tasks can therefore be accomplished by algorithms where clarity is not important and Neural Networks, a typically hard to explain technique, can be used without issues.

A different case is Google Deepmind's *Alpha Zero* chess engine. This engine represents today the benchmark for playing chess and outsmarts any human or traditional chess engines by a margin[1]. It is the end-result of a 3-year long project with hundreds of the smartest minds of the planet and one can imagine that the stakes on the performance of this algorithm are very high. However, this project does not represent an example of explainability importance. This is mainly due to the fact that the performance assessment can easily be made without the need for explanation: there is historical evidence (that is, hundreds of thousands of played games that the algorithm is performing well and playing close-to-prefect games in any position. Moreover, the quest for better understanding such a "magical algorithms" interests a community limited to the elite scientific

[1] Alpha Zero's Elo rating is estimated to be around 3400, compared to top Grandmasters' Elo of 2800

intellectuals and the chess fan-base. Additionally, since such failures are not happening, we are witnessing a vibrant chess community where amateur fans and masters are happy to interact with such chess-playing engines and are happy to discover unpredictable (but correct) reactions of the algorithm: it does not matter if that engine's choice will be understood only a couple of moves later. We conclude that an explanation of the outcomes of the algorithm would be desirable but is not necessary.

As a third example we can cite YouTube's newest automatic filter algorithm against hate speech (YouTube®, 2019). Again, it is an algorithm based on ML techniques which classifies and flags videos which do not respect the platform's policy. The true performance of this algorithm cannot proven by any "historical" data and it is prone to spectacular errors[2]. However such losses are acceptable from YouTube's own perspective as a private company, as what is most important to them is that the algorithm's predicting power is above a certain threshold. Although such example raises extremely important ethical issues, it has little impact on the revenues of the U.S. tech giant and not-explainable ML algorithms can be used since at the moment they are the best available option.

It is worth mentioning that in both last two examples, explainability is not achievable at present state. In the game of chess for example, there are criteria such as "value" and "activity" of the pieces which are important for good human chess players to evaluate a game's position and the best move to make next. However, these factors are not measurable objectively, nor unanimously, and they are developed in a personal fashion through studying and (lots of) experience from past games. Machine Learning algorithms also learn from a huge database of games, however the patterns they make use of in the data can be very different from the human ones. Human-friendly notions such as "activity" or "value" might have no Machine Learning counterpart, making the explainability of the algorithms at the current state not feasible.

On the other hand, a Credit-Risk model has both ingredients to boost the importance of explanatory techniques: it has no guaranteed historical evidence on its performance and the stakes are high in case of unexpected behaviour. The reason behind the lack of historical evidence lies on the fact that the format of collected data in credit institutions can vary in time: market behaviour changes a lot in periods of time as short as five years, new regulations appear and data formats and collection change, either to fulfil the new requirements or for organisational purposes. Moreover, some variables which are important in the final result can be hidden to human understanding and current models, either because of being too complex to grasp, or because they have not been observed historically yet. Such variables are usually called *unknown unknowns* as in Jorion (2009).

These four examples are resumed in Table 1.1. As we can see the banking side is a perfect example to show the importance of explainability. Stakes are high and there is no lasting experience with the application of such techniques to the field to guarantee the robustness of the algorithms under the ever-changing economy or under a cracking crisis. We conclude that for such models explainability is important and techniques like Neural Networks cannot be safely adopted at the present state. Instead, more accessible techniques such as Decision Trees, Gradient Boosting or Random Forests

[2]One example which caught the attention of the media is: https://www.buzzfeednews.com/article/ryanhatesthis/history-teacher-scott-allsop-youtube-channel-banned-nazi

can be implemented provided that the explanatory techniques in use are good enough and are able to bridge the gap between the engineering and the business side of the corporation.

	Low stakes	High stakes
Historical data on performance	Handwriting Recognition	Chess engines
No historical data on performance	Video classification	Credit Risk modelling

TABLE 1.1: Importance of explainability of models, based on the presence or absence of historical data and on the stakes in case of failure. In green: little need for explainability. In yellow: explainability is desirable. In orange: explainability is extremely important.

For this reason, ABN Amro is currently trying to improve the explanability of its Machine Learning-based models, and the motivations are described in the internal documentation (Gemmell, 2018). The main points can be summarised as the following:

- *Building trust* in Machine Learning models: models should be trusted by the department they are used in, once a business experts understand how a model makes a prediction, then they are able to make an informed decision and used the model at its best and full potential. We shall aim to make tools which are useful and controllable by humans, rather than the other way round.
- *Improve* Machine Learning models: if business experts understand the models developed by the Machine Learning Team, then they might be able to suggest improvements. For example, they can suggest using new variables which are believed to have a strong influence on the outcome, or they can recover more useful data via their own network and contacts.
- *Identify risks and limits* of Machine Learning models: one the main issue for such algorithms consists in overestimating their performance and predicting power. This latter issue cannot be tackled directly by the project developers, but if modellers can explain the operations within their "black boxes" to a business expert, the latter can identify whether some data has been used inappropriately or whether the outputs are unreasonable in certain situations.

A credit risk modelling framework best shows the importance of explainability, however its importance is appreciated well beyond a corporation's interest: it reached the academia decades ago already, and still is a flourishing field for publications with articles such as Ribeiro et al. (2016) proving that explainability enhances trust from the user's side. The European Commission (2019) is also pushing for a similar direction by publishing a set of guidelines from a high-level expert group on AI set up by the Commission, which identified four ethical principles for an AI to be trustworthy. Among the principles we cite fairness and explicability, and according to the guidelines these

principles can be pursued by seven key requirements, among which we cite transparency and non-discrimination. These requirements are particularly sensible and we will test the ML algorithms we use in this work to check whether they fulfil these principles.

1.2 Plan of the Thesis

To achieve the purpose, we will implement the Partial Dependency (PD) Plot technique introduced by Friedman (2001) and its refinement called Individual Conditional Expectation (ICE) Plots by Goldstein et al. (2015). These are some of the many tools which have been developed in recent years to help human users understand the outcomes of machine-learning algorithm, we will apply and further refine these techniques in two ML models, the first one being a result of individual initiative based on the Boston Housing Dataset, the second one being the FR&R Loss Assessment Grade (FLAG) model, currently in use at ABN Amro. The accumulated on-field experience will suggest the creation of new variants of these tools, namely the d-ICE flatness detector in Section 2.7.5 and the log-d-ICE in Section 3.1.4. We will also link the latter concept with the statistical concept of dispersion.

Chapter 2

Methodology

As we mentioned before, the process of "learning" for such algorithms begins with observations or data in order to look for patterns and make better decisions withe the future examples and data that are provided. The primary goal for machine-learning is to allow the computers learn such patterns without human intervention, and make their predictions based on future data accordingly to what has been learned in the past.

The data usually consists in a set of observations or examples, each containing one or more entries. The set of entries representing the actual data and knowledge we have concerning an observation is called *domain set* and is indicated with the letter $\mathcal{X}$, while those entries which the machine learning algorithm is supposed to learn how to predict in the future samples is called *label set* and is indicated with $\mathcal{Y}$. All such entries may contain numerical values (e.g. physical measurements of an object), strings of text, or they can be empty. These entries can be either continuous, discrete or categorical. Each of these entries is called *feature*. We will indicate with p the number of features in the domain set and with q the number of features in the label set. Without loss of generalisation we can assume all features lie in $\mathbb{R}$: if they don't, we can extend the domain (in the case of the categorical and discrete features), or map the features to the real numbers (as in the case of strings of text) so that they are included. To be more formal we can follow the definitions from Shalev-Shwartz and S. Ben-David (2014, pg. 13) and say:

Definition (Training Data)**.** The *training data*, usually denoted as S, is a finite sequence of pairs of vectors in $\mathbb{R}^d \times \mathbb{R}^q \ni \mathcal{X} \times \mathcal{Y} : \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_m, \boldsymbol{y}_m)\}$. It represents a sequence of labelled domain points. This is the input that the learner has access to.

We denote vectors in bold, therefore $\boldsymbol{x}_i \in \mathbb{R}^d$ stands for the i-th observed datapoint, and the notation $x_{i,j}$ indicates the j-th component of the i-th observation. In case of the feature space, in this work we will consider the case $q = 1$ alone, hence we assume $\mathcal{Y} = \mathbb{R}$ and the $\boldsymbol{y}_i$-s will we refer to value of the i-th observed label, the bold notation for $\boldsymbol{y}_i$ will be dropped from now on.

The key principle pf Machine Learning is to build models of a real-world situation based on data. Models are a simplified description of reality, and a good model ought to be structured enough to satisfactorily replicate reality but simple enough to be replicable. In our context, we will consider

models as elements of the set $\mathcal{Y}^{\mathcal{X}}$, that is the set of functions from $\mathcal{X}$ to $\mathcal{Y}$. By doing so, we We can split machine learning models in two groups according to the structure of the label set $\mathcal{Y}$:

Definition (Regression model)**.** Given the domain set $\mathcal{X}$ and a label set $\mathcal{Y}$ a *regression model* consists in a function $f : \mathcal{X} \to \mathcal{Y} = \mathbb{R}$.

Any regression model f can be seen as a mapping between a d-dimensional feature space $\mathcal{X} \in \mathbb{R}^d$ and and a prediction scale $\hat{y} \in \mathbb{R}$. $\hat{y} = f_r(X)$ where X represents the infinite set of points in the feature space. Possible regression algorithms are Linear Regression, Logistic Regression, Polynomial Regression, Lasso Regression and Random Forest. Regression tasks include models such as house pricing, risk assessment and so on.

Other problems are not suitable for a regression model. This happens where the set of possible outcomes (labels) is small and there is no underlying structure between such labels. That is where there is no notion of "closeness" (or distance) between labels, nor mathematical operations among them make sense. For example in classification problems with 5 possible labels, a datapoint labelled as "type 2" does not necessarily have anything to do with the average between item " type 1" and item "type 3"; nor item "type 1" necessarily shares more features in common with a "type 2" one rather than with a "type 5". Whee this kind of situation happens, the labelling problem is part of the so called *classification* tasks, and classification models are developed for this scope. Formally a classification model is:

Definition (Classification model)**.** Given a model space $\mathcal{X}$ and a set of labels $C = \{c_1, ..., c_k\}$ *classification model* consists in a function $f : \mathcal{X} \to C$.

Note that we can easily map C to $\mathbb{R}$, but we stress again the fact that no structure underlying $\mathbb{R}$ such as the notion of distance and order should be inherited to C unless also C has it. Classification algorithms take care of this aspect, and a few examples are: Logistic Regression, Nearest Neighbour, Decision Trees and Random Forest. Once the data is collected and the learner uses them to build an output function, we can say we have obtained a:

Definition (Learner's output)**.** It consists in a prediction rule $h : \mathcal{X} \to \mathcal{Y}$. Such function is also called a predictor, a hypothesis, or a classifier. Given a sample S, we will denote the set of predicted labels $\{\hat{y}_1, \ldots, \hat{y}_n\}$ by $\hat{\mathcal{Y}}$. The learner's output can be used to predict the label of new domain points.

It is worth noting that $\hat{\mathcal{Y}}$ does not need to be the same set as $\mathcal{Y}$, but since all their elements in real-world models will be represented in floating-point arithmetic we shall assume $\hat{\mathcal{Y}} = \mathcal{Y}$. Also worth noting is that the choice of the learner can be any function of the set $\mathcal{Y}^{\mathcal{X}}$, but in practice the set can be much smaller (and finite) for implementation constraints. This leads us to define:

Definition (Hypothesis class)**.** Is denoted by $\mathcal{H}$ and consists in a predetermined class of predictors $\mathcal{H} \ni h_i : \mathcal{X} \to \mathcal{Y}$. In a machine learning setting the algorithm terminates by selecting a predictor h_i after "learning" from the data. It holds that $\mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$.

Given the set $\mathcal{H}$, we need a mean of comparison so that the algorithm can *choose* the best learner according to some metric. In order to do this we introduce the concept of *loss* associated to a

prediction rule. That is, we define a function mapping events, or outputs, onto a real number. Such number intuitively represents the "cost" associated with the event happening, and the related function is the *loss function*.

2.1 Loss functions

The mathematical definition of loss function can vary a lot depending on the type of machine learning algorithm, and since an all-embracing definition would be vague and inconclusive, we prefer to build such definition within a narrower scope where an intuitive definition can be given. Our choice falls on a definition for loss function applicable to supervised learning algorithms only. Such choice is dictated by the fact that all algorithms we will be using in this chapter and the following one will all be supervised learning algorithms. The definition of supervised learning, if not already known to the reader because of his/her background, will be given later on.

Definition (Loss function)**.** A *loss function* is a function that maps the algorithms output onto a real number. In case of supervised learning algorithm we define it as a function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. Given a prediction $\hat{y}_i$ and the true output y_i, the loss function is written as $\ell(y_i, \hat{y}_i)$. The loss function intuitively represents a cost associated with the event of being predicted the value $\hat{y}_i$ instead of its true value y_i.

Usually loss functions are defined so that $\ell(y_i, y_i) = 0$, in other terms the loss incurred with an exact prediction is usually zero. Examples of loss functions are:

- **p-norm Loss:** these are loss functions of the kind $\ell(y_i, \hat{y}_i) = \alpha|y_i - \hat{y}_i|^p$, with $k > 0$ and $p \in \mathbb{N} \setminus 0$. The most common ones have $k = 1$ and are the absolute-difference loss function ($p = 1$) and the squared-loss function ($p = 2$). These losses are suitable for regression problems and we will refer to them as L_1 and L_2 respectively.

- **Huber Loss:** it is a piece-wise defined function. Given a threshold δ, the Huber loss is defined as:

$$L_\delta(\hat{y}_i, y_i) = \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2 & \text{if } |y_i - \hat{y}_i| \leq \delta, \\ \delta\,|y_i - \hat{y}_i| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

  The Huber loss has a mixed nature: it behaves like a L_2 norm for small values of $|y_i - \hat{y}_i|$ but linearly for large values. This mixed behaviour allows the Huber loss to combine the sensitivity of the minimum-variance estimator of the mean (from the quadratic loss function) with the robustness of the absolute value function. Because of these desirable properties, we will use the Huber loss in our application in the Boston Housing Dataset in Chapter 3. The main drawback of the Huber loss is that it is continuously differentiable, but its derivative is not (that is, Huber loss functions lies in C^1 but not C^2). Since some machine learning algorithms need the second derivative of the loss, a twice-differentiable loss function can be preferred.

- **0-1 loss:** it is defined as $L_p(y_i, \hat{y}_i) = \mathbf{1}\{\hat{y}_i \neq y_i\}$, and is typically used in classification problems. Note that unlike the previous losses, the $0-1$ loss is not differentiable, nor convex.

These losses are defined over a single prediction $\hat{y}_i$ opposed to the true value y_i, but we can extend the definition to the whole dataset. Given a sample S and a predictor h, we have an associated $\hat{\mathcal{Y}}$ from which we can define the set of residuals $\{r_1, \ldots, r_n\}$ defined as $\{y_1 - \hat{y}_1, \ldots y_n - \hat{y}_n\}$. This corresponds to the set of differences between the true and the predicted value. The accumulated (or more commonly, the average) loss over the sample S is an indicator of performance of the learned model and is also used during the learning phase of the ML algorithm. Choosing suitable loss functions for machine learning algorithms is also important, and here we introduce some commonly used ones:

- **Mean Squared Error Loss:** referred as MSE, is linked to the the L_2 loss, is defined as the average 2-squared-norm loss incurred in the analysed dataset $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ when predicting $\{\hat{y}_1, \ldots, \hat{y}_n\}$ instead:

$$\mathrm{MSE}(S) = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 . \tag{2.1}$$

- **Mean Absolute Error Loss:** over a sample S is a close relative of the L_1 loss. It is defined as:

$$\mathrm{MAE}(S) = \frac{1}{n} \sum_{i=1}^{n} |\hat{y}_i - y_i|^2. \tag{2.2}$$

- **Huber Loss:** for a dataset S is defined as the average of the incurred Huber losses over the observations:

$$\mathrm{HUB}_\delta(S) = \frac{1}{n} \sum_{i=1}^{n} \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2 & \text{if } |y_i - \hat{y}_i| \leq \delta, \\ \delta\,|y_i - \hat{y}_i| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

  The δ parameter is tuned according to the available dataset. Usually the goal is to make most of the data fall within the quadratic behaviour interval in $[-\delta, +\delta]$, and outliers outside. In order to achieve this, the value of δ is usually set as a top percentile (e.g. 5%) of the absolute value of the residuals $r_i := y_i - \hat{y}_i$. An alternative approach leading to the same result is to scale the residuals so that the absolute value of the (5th) top percentile is equal to 1, and then use the Huber loss with $\delta = 1$. This approach can be applied to a wider context, and we will see that is also works with the next loss function.

- **Log-cosh loss:** is a function used in regression tasks, similar but smoother than HUB. Log-cosh is the logarithm of the hyperbolic cosine of the prediction error and the loss is defined as:

$$L_{\cosh}(S) = \frac{1}{n} \sum_{i=1}^{n} \log\left(\cosh(r_i)\right) = \frac{1}{n} \sum_{i=1}^{n} \log\left(\frac{e^{r_i} + e^{-r_i}}{2}\right) \tag{2.3}$$

We can easily verify that as a residual $r_i \to \infty$, the log-cosh loss asymptotically tends to $r_i - \log(2)$. As $r_i \to 0$, Taylor expansion shows a behaviour like $r_i^2/2$. This means that such loss function has a L_1 behaviour for large residuals, and a quadratic one for small ones, just like the Huber loss. We can appreciate how these losses compare to each other through Figure 2.1.

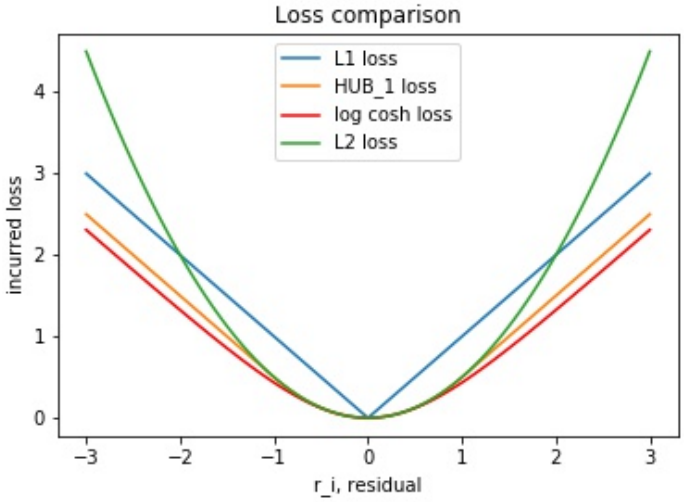

FIGURE 2.1: Comparison of the introduced losses. Both the Huber with $\delta = 1$ and the log cosh loss have a similar behaviour compared to the L_2 loss multiplied by a factor 0.5. The latter still has stands out as the biggest incurring loss when $r_i > 2$.

The advantage of this loss over the Huber loss is that the $L_{\cosh}$ is an analytic function and therefore twice differentiable. This is a remarkable advantage as some ML algorithms may use the second derivative of the loss functions in the learning process, and are not able to do so when using Huber loss. A drawback of the the log-cosh consists in its tuning, being less intuitive than the Huber one. This is reason why the alternative tuning for the Huber loss was suggested: we can use the same normalising principle for the log cosh-loss. To see in details how, we make the following quantitative remark:

Remark. If we compare asymptotic behaviour of Log-cosh and Huber loss, we realise that:

$$|\mathrm{HUB}_1(y_i - \hat{y}_i) - L_c(y_i - \hat{y}_i)| \text{ is bounded by } \quad \log(2) - 0.5 \approx 0.2 \tag{2.4}$$

This, added to the fact that they tend to have the same values in a neighbourhood of zero, leads us to conclude that we can bound the difference with a (small) constant ≈ 0.2.

This suggests that, multiplying the residuals by a factor α such most of the new $\{\alpha r_1, \dots \alpha r_n\}$ lie in the $[-1, +1]$ interval is an effective way to use the log-cosh loss function. It follows that $\alpha = 1/\delta$ from the way we tuned δ in the Huber loss.

The comparison of the most used loss functions is now complete. We intentionally spent some pages (and time) on this topic because of the importance of loss functions in machine learning. These loss functions indeed play two important roles in the process of learning:

- Firstly, evaluating the loss incurred by a learned model gives a mean of comparison between models: those which at the end of the training process incurs in lower losses are expected to be more accurate than trained models incurring in greater loss. Losses are the basis for performance metrics in machine learning.
- Secondly, machine learning algorithms are structured in such a way that after every iteration the loss of the learner is calculated. The following step, modification to the prediction rule are made so that the loss can be decreased. Loss functions therefore play a key role during the training phase of ML algorithms as well.

2.2 True error and empirical error

Once we have chosen a loss function, we shall use it to assess the performance of a learner. To do so, we assume that each pair in the training data is generated by first sampling a point x_i from and underlying distribution $\mathcal{D}$ and then labelling it via the labelling function f. We keep following the notation from Shalev-Shwartz and S. Ben-David (2014), and we now define the error of a prediction rule $h : \mathcal{X} \to \mathcal{Y}$ as:

$$L_{\mathcal{D}}(h) = \mathbb{E}_{\mathcal{D}}[\ell(y, h(x))]. \tag{2.5}$$

Where the error of h is the average loss incurred when randomly choosing an example x from the distribution $\mathcal{D}$. Such $L_{(\mathcal{D})}(h)$ has several synonymous names such as *generalisation error*, *risk*, or *true error* of h.

A problem arises when the learning algorithm does not know the underlying distribution $\mathcal{D}$, nor the labelling rule f. In this case, the true error is not directly available to the learner ad no minimisation is therefore possible. To overcome this, a useful approximation can be used, the so called *training error*. This training error, also called *empirical error* or *empirical risk* measures the average loss incurred by the learner in the available sample S. In formulas:

$$L_S(h) = \mathbb{E}_S[\ell(y, h(x))]. \tag{2.6}$$

When the algorithm/learner has to make the choice, the most natural one will be the one minimising the empirical risk over the sample S. We call such optimal learner the *Empirical Risk Minimiser* (ERM):

$$\mathrm{ERM}_{\mathcal{H}}(S) \in \underset{h \in \mathcal{H}}{\arg\min}\, L_S(h). \tag{2.7}$$

Introducing this estimate of the true error is not always a good idea as there are cases when $L_S(h)$ is arbitrarily small but the true risk $L_{\mathcal{D}(h)}$ is not. This phenomenon is called *overfitting*. The opposite problem can also occur when the model has poor predictive performance with respect to a benchmark because of being too simple and failing to capture the underlying trend of the data. In this case we have $\mathrm{ERM}_{\mathcal{H}}(S) > \epsilon$, where epsilon is our benchmark, and we call this phenomenon *underfitting*.

A simple explanatory example

To better explain the issue we provide an example with a regression problem. Despite being a simple problem which requires nothing more than a polynomial fit, it is complex enough to raise some issues and give ideas about how to tackle the problem. The example consists on the following set up. Consider 25 observations uniformly drawn from $\mathcal{X} = [0, 1] \in \mathbb{R}$ and let the observed y_i be generated following the rule:

$$Y = (X^3 + 3\,X^2 - 4\,X + 3) + \mathcal{E}, \qquad X \overset{iid}{\sim} \mathrm{Unif}(-1, 1) \qquad (2.8)$$
$$\mathcal{E} \overset{iid}{\sim} \mathcal{N}(0, 0.1^2)$$

that is, the underlying labelling function is a cubic polynomial $f(x)$ with some added noise $\mathcal{E}$ is distributed as $\mathcal{N}(0, 0.1^2)$[1]. However, the true underlying polynomial function is unknown to the learner, thus various polynomial fits of degree 1, 4 and 10 are comapred in Figure 2.2. The outcomes are shown in red, the true underlying pattern $y = f(x)$ cleared of noise is plotted with a dashed green line, and the true observed datapoints (with noise) are shown in blue. To assess the performance of the fits, the MSE loss will be used, and the performance on a sample S is defined as $1 - \mathrm{MSE}(S)$.

To the left in Figure 2.2 it is shown what underfitting looks like in a regression problem. An underfitted learner is not able to capture the complexity of the data: the approximately parabolic behaviour is seen as a decreasing linear trend. As a consequence the predictive performance of the algorithm is "low" $(1 - \mathrm{MSE}(S) = 0.874)$ for observed data and one does not expect the algorithm to perform well for future data either.

The rightmost picture shows the opposite problem, that is overfitting. The degree of the fitted polynomial is too high, as a result the model closely fits the observed data, but also fits the added noise when doing so. As a result, we obtain a predictor with wobbling behaviour which does not match the true underlying model, especially in regions such as the border of the domain. If we were to judge the performance of the fit based on present data we would get very good results $(1 - \mathrm{MSE}(S) = 0.993)$, but if we were to measure the performance on new data we would not be able to achieve the same performance. The reason for this happening lies on the fact that when we raise the degree of the polynomial fit, we are giving a higher number of degrees of freedom to the learner, and we expect the model to better fit the given data. However we cannot achieve the

[1] We use the $\mathcal{N}(\mu, \sigma^2)$ notation throughout this work

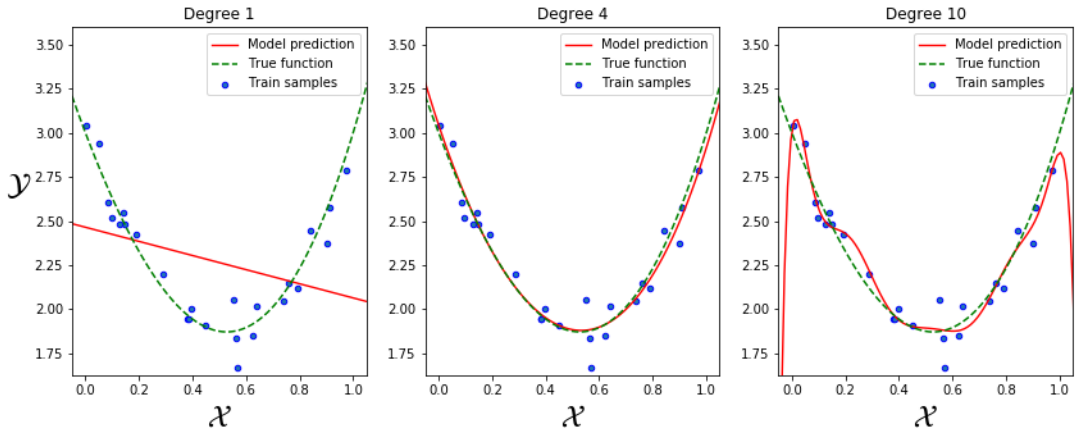

FIGURE 2.2: Regression problem case with polynomial fitting. To the left: example of underfitting. In the middle: good fit. To the right: overfitted model.

same increase with respect to the future data: new x_i-s and related y_i-s will have independent noise from the previous observations, and the the model will not be fitting this noise anymore. The performance will sensibly drop for future data and we will be far from the one attained during the training process, this problem is commonly referred as lack of *generalisation*. Overfitted models are not desirable because of their unexpected behaviour and overestimated performance.

The middle picture shows a good compromise between the previous models. The fitting polynomial has degree 4 (comparable to the true underlying model) and the predictive power is good ($1 - \text{MSE}(S) = 0.984$) mainly limited by the presence of noise, with the latter not being wrongfully fitted by the polynomial. The red and green curves lie close to each other in the whole domain, that is the model's predictive power will be good for future data sampled from the same $(\mathcal{X}, \mathcal{Y})$. We therefore expect the future performance of the model to be comparable to the one attained during the training phase, that is, we expect the model to be able to generalise. This is a very important requirement for a model, as the true indicator of performance of Machine Learning lies in its predictive power.

2.3 Preventing under and overfitting

From the previous example we discovered that empirical loss $L_{S,f}$ can significantly differ from the "true" loss $L_{\mathcal{D},f}$, and problematic is the case when the former is small but the latter is not. In particular, we are worried that future data might not fit with the predicted model especially if such

data is slightly perturbed with respect to the original one, as it can be the case of time-depend data in many real-world problems.

A solution to these issues is presented in this Section, and a connection with the theoretical framework introduced between pages 7 and 12 is found. First of all, a useful tip is to split the available dataset S into a *train set* S and a *test set* T. That is, we only use the fist set to train data (even if this means losing some performance because of the smaller amount amount of data), and then we calculate the incurred loss on the train set. Typically the train set includes $80\% - 90\%$ of the data, and the remaining represents the test set. The test set will be an imitation of the "future data" we were being worried about in our example. While loss on the train set corresponds to the empirical loss $L_{S,h}$, the loss incurred on the test set $L_{T,h}$ is a proxy for the true loss $L_{\mathcal{D},h}$. Thanks to this trick, we can now tell whether a model overfits simply by comparing the performance on the training and testing set: if the latter is sensibly lower, then the model is overfitting. Deciding how "big" the difference should be is up to the user, to the context, and depends on the adopted algorithm. In order to face this, we generate 1000 new datapoints from $\bar{\mathcal{X}} = [-1.025, +1.025] \in \mathbb{R}$, a slightly perturbed version of the original $\mathcal{X}$ and we use these points as a test set. The incurred loss $L_{T,h}$ is calculated. In this example the test set will be larger than the train one but this anomaly is justified by the fact that more testing datapoints lead to a more accurate estimate of the accuracy, and $L_{T,h}$, will be a more accurate estimator for $L_{\mathcal{D},h}$. Usually, data cannot be generated at will as in this case and the precision for the performance of the training and testing set is limited by the amount of data. Moreover train and test performance metric can be sensitive to how the splitting between train and test data is made. To mitigate this issue, *K-fold cross-validation* is used, but we will describe this technique in more detail later in this work (see Section 2.4.1).

Thanks to this train-test split procedure, we are now able to find which polynomila fit is best for the data from the process in (2.8). We do this by applying the procedure to all possible polynomial fits of a given degree, up to 19. That is, we calculate the incurred loss for the train and test set for every polynomial fitting with degree varying between 1 and 19. Let k be the degree of the polynomial, the algorithm searches for the best polynomial fit of degree (at most) k and calculates the incurred loss in both the training and testing set, and the results are shown in Figure 2.3:

2.3.1 Tentative definition of underfitting and overfitting

In the following paragraphs, we give some mathematical foundations to the intuition of underfitting and overfitting shown in Figure 2.2. Let us set a lower benchmark L_B and an upper benchmark L_U, and call $\bar{\varepsilon}$ their difference. We say the algorithm is:

- *underfitting* when both the train and test performances fall below the lower benchmark L_B
- *good fit* when both the train and test performances are above the lower benchmark L_B, and their difference is less than $\bar{\varepsilon}$.
- *overfitting* when the train performance is above L_U and the diference between train and test score is above $\bar{\varepsilon}$.

Setting L_B is a somewhat arbitrary choice and we decide to set it to $L_B = 0.975$. Setting the upper benchmark on the other side, can be done based on the fact that any model scoring *better* than the true underlying model f is overfitting. We can then calculate the performance for f and set its performance as the upper bound L_U. Calculating he loss of f is straightforward:

$$\begin{aligned} L_{\mathcal{D},f}(f) =& \mathbb{E}_{\mathcal{D},f}\left[\ell(Y, f(X)\right] = \mathbb{E}_{\mathcal{D},f}\left[(Y - f(X))^2\right] = \\ =& \mathbb{E}_{\mathcal{D},f}\left[(f(X) + \mathcal{E} - f(X))^2\right] = \mathbb{E}_{\mathcal{D},f}\left[\mathcal{E}^2\right] = 0.01. \end{aligned} \tag{2.9}$$

Which means that the performance is 0.99. We set $L_U = 0.99$, and show it in Figure 2.3 with the dotted red line.

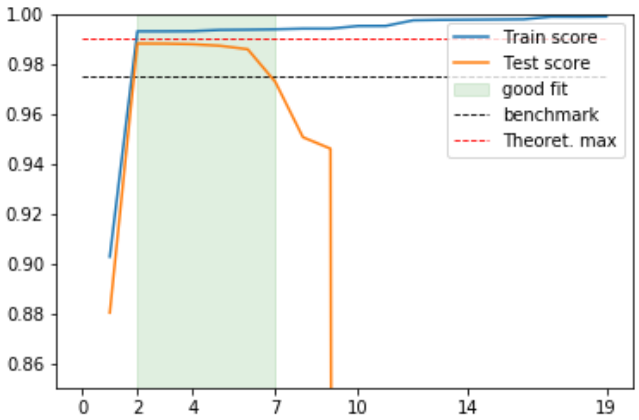

FIGURE 2.3: Train and test score as the maximum degree of the polynomial fit varies. The "sweet spot" for fitting is between 2 and 6 as these are the values for which the test score is higher. A 7th-degree fit also meets the benchmark but already shows some overfitting.

Starting from the rightmost side of Figure 2.3, we see that the performance of a linear regression is poor as it does not capture the parabolic-like trend of the data. Both train and test score are below the threshold, meaning there is underfitting. A fit of degree 2 is already good enough for this purpose even if the underlying $f(x)$ has degree 3, and a similar performance level is maintained for all fits up to $k = 6$. Starting from $k = 7$ the test score starts decreasing as the train score keeps increasing and we witness overfitting: the trained polynomial are fitting the 25 datapoints "too well" and the undesired wobbling behaviour shown in the rightmost picture of Figure 2.2 appears if we investigate the output of the model. In the extreme case of a polynomial fitting of degree 24, we would witness a *perfect* fit with the training data, and a poor performance on the testing.

We conclude that for our case good polynomial fits are those with degree $k \in \{2, 3, 4, 5, 6\}$. Having to choose among these, a good rule is to choose the simplest model, that is the parabolic fitting. But how an we put this into practice for a true machine learning model, where no polynomial fits are present (and maybe not even effective in capturing the underlying patterns)? The answer is relatively simple if we interpret the research of the best polynomial fit k as a parameter tuning process:

Remark. This idea applies more generally when fitting model hyper-parameters, and choosing the degree of the polynomial fitting can be considered a particular case of hyper-parameter tuning. When a parameter of a ML algorithm needs to be tuned, a GridSearchCV can be done, command available in Python within the sklearn library.

The polynomial degree fitting is also related to an interesting theoretical turn, we can indeed interpret the degree tuning as a hypothesis class restriction, and restricting the learner to choosing a predictor from a smaller $\mathcal{H}$, is equivalent to biasing towards a particular set of predictors according to Shalev-Shwartz and S. Ben-David (2014). Since the choice of such a restriction is determined before the learner sees the training data, it should ideally be based on some prior knowledge about the problem to be learned. On the opposing side, restricting the hypothesis class too much leads to underfitting. In our example, the three stages of fitting can be represented as different hypothesis classes of increasing size. In particular:

$$\mathcal{H}_1 = \{a_0 + a_1\, x : a_0, a_1 \in \mathbb{R}\} \tag{2.10}$$

is the smallest class, and leads to underfitting. The middle case is represented by:

$$\mathcal{H}_4 = \left\{ \sum_{i=0}^{4} a_i\, x^i : a_i \in \mathbb{R},\ \forall i \leq 4 \right\}, \tag{2.11}$$

while overfitting happens when the a-priori selected class is too big, such as in:

$$\mathcal{H}_{10} = \left\{ \sum_{i=0}^{10} a_i\, x^i : a_i \in \mathbb{R},\ \forall i \leq 10 \right\}. \tag{2.12}$$

Now, the formerly introduced classes are infinite and isomorphic to $\mathbb{R}^2, \mathbb{R}^5$ and $\mathbb{R}^{11}$ respectively. Yet, in practical applications, these classes are finite as the floating-point representation of each coefficient is limited to 2^{64} possible values[2]. Limiting the learner to prediction rules within some finite hypothesis class can therefore be considered as a mild restriction, and leads to interesting results as we will see in the following paragraphs. In particular, theory about the so called *PAC learnability* gives interesting bounds and insight about overfitting. We think is worth introducing these concepts in the following Section.

[2]This means that $\mathcal{H}_k$ has cardinality $2^{64(k+1)}$, for k positive integer.

2.3.2 PAC Learnability

The section is made of two parts: firstly we show that if $\mathcal{H}$ is a finite class, then the $\text{ERM}_{\mathcal{H}}$ rule will not overfit provided it is based on a sufficiently large training sample; finally, we introduce the concept of Probably Almost Correct (PAC) learnability in machine learning. Throughout this Section we largely rely (and keep the notation[3]) from Shalev-Shwartz and S. Ben-David (2014, p.17-27).

To start with, we analyse the performance of the $\text{ERM}_{\mathcal{H}}$ learning rule assuming that $\mathcal{H}$ is a finite class. For a training sample S, labelled according to some $f : X \to Y$, we denote by h_S the result of applying $\text{ERM}_{\mathcal{H}}$ to S, namely,

$$h_S \in \underset{h \in \mathcal{H}}{\arg\min}\, L_S(h). \tag{2.13}$$

Now we make the following simplifying assumption, which will be relaxed soon:

Definition (The Realisability Assumption)**.** . There exists $h^* \in \mathcal{H}$ s.t. $L_{\mathcal{D}}(h^*) = 0$. Note that this assumption implies that with probability 1 over random samples S, where the instances of S are sampled according to D and are labelled by f, we have $L_S(h^*) = 0$.

The realisability assumption implies that for every ERM hypothesis we have that $L_S(h_S) = 0$ with probability 1. However, we are interested in the true risk of h_S, $L_{\mathcal{D}}(h_S)$, rather than its empirical risk. Clearly, any guarantee on the error with respect to the underlying distribution $\mathcal{D}$ for an algorithm that has access only to a sample S should depend on the relationship between $\mathcal{D}$ and S. The common assumption in statistical machine learning is that the training sample S is generated by sampling points from the distribution $\mathcal{D}$ independently of each other. Formally: the examples in the training set are independently and identically distributed (i.i.d.) according to the distribution $\mathcal{D}$.

Now, since $L_{\mathcal{D}}(h_S)$ depends on the training set S and that training set is picked by a random process, there is randomness in the choice of the predictor h_S and consequently $L_{\mathcal{D}}(h_S)$ is a random variable. It is not therefore realistic to expect that with full certainty that S will suffice to direct the learner toward a good classifier (from the point of view of $\mathcal{D}$), as there is always some probability that the sampled training data happens to be non-representative of the underlying distribution $\mathcal{D}$. We will therefore address the probability to sample a training set for which $L_{\mathcal{D}}(h_S)$ is not too large. Usually, we denote the probability of getting a non-representative sample by δ, and call $(1 - \delta)$ the confidence parameter of our prediction. Moreover, the parameter ϵ introduced in page as benchmark to beat, will be called *accuracy parameter*. We interpret the event $\{L_{\mathcal{D}}(h_S) > \epsilon\}$ as a failure of the learner, and we are interested in upper bounding the probability to sample m-tuple of instances that leads to failure of the learner. Let by $\mathcal{D}^m$ denote the (probability) distribution over m-tuples induced by applying $\mathcal{D}$ to pick each element of the tuple independently of the other members of the tuple. $S = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_m, \boldsymbol{y}_m)\}$ be the instances of the training set. We would like to upper bound:

$$\mathcal{D}^m(\{S : L_{(\mathcal{D},f)}(h_S) > \epsilon). \tag{2.14}$$

[3]From now on, we drop the labelling function f when referring to the losses L_S and $L_{\mathcal{D}}$

Now, let $\mathcal{H}_B$ be the set of "bad" hypotheses, that is the set of learners that incur in a loss greater than ϵ:

$$\mathcal{H}_B = \{h \in H : L_{(\mathcal{D},f)}(h) > \epsilon\}. \tag{2.15}$$

In addition, let

$$M = \{S : \exists h \in \mathcal{H}_B, L_S(h) = 0\} \tag{2.16}$$

be the set of misleading samples: namely, for every $S \in M$, there is a "bad" hypothesis $h \in \mathcal{H}_B$ that looks like a "good" hypothesis on S. Now, recall that we would like to bound the probability of the event $L_{(\mathcal{D},f)}(h_S) > \epsilon$. But, since the realisability assumption implies that $L_S(h_S) = 0$, it follows that the event $L_{(\mathcal{D},f)}(h_S) > \epsilon$ can only happen if for some $h \in \mathcal{H}_B$ we have $L_S(h) = 0$. In other words, this event will only happen if our sample is in the set of misleading samples M. Formally, we have shown that:

$$\{S : L_{(\mathcal{D},f)}(h_S) > \epsilon\} \subseteq M. \tag{2.17}$$

Now, note that we can rewrite M as

$$M = \bigcup_{h \in \mathcal{H}_B} \{S|x : L_S(h) = 0\}. \tag{2.18}$$

Hence,

$$\begin{aligned} \mathcal{D}^m\left(\{S : L_{\mathcal{D}}(h_S) > \epsilon\}\right) \leq & \mathcal{D}^m(M) = \\ = \mathcal{D}^m\left(\bigcup_{h \in \mathcal{H}_B} \leq \{S : L_S(h) = 0\}\right) & \leq \sum_{h \in \mathcal{H}_B} \mathcal{D}^m(S : L_S(h) = 0). \end{aligned} \tag{2.19}$$

Where the last inequality follows from applying the union bound Lemma (see Theorem A.2.1). This inequality bounds the probability of incurring in a loss greater then ϵ (following the ERM rule) by the sum of probabilities that a learner incurs in no empirical loss, over all possible elements of the bad hypotheses set $\mathcal{H}_B$. Next, let us bound each summation of the right-hand side of the preceding inequality. Fix some "bad" hypothesis $h \in \mathcal{H}_B$. The event $L_S(h) = 0$ is equivalent to the event $\{\forall i, h(\boldsymbol{x}_i) = f(\boldsymbol{x}_i)\}$. Since the examples in the training set are sampled i.i.d. we get that

$$\mathcal{D}^m(S|x : L_S(h) = 0) = \mathcal{D}^m(\{S|x : \forall i, h(\boldsymbol{x}_i) = f(\boldsymbol{x}_i)\}) = \prod_{i=1}^{m} \mathcal{D}(\{\boldsymbol{x}_i : h(\boldsymbol{x}_i) = f(\boldsymbol{x}_i)\}). \tag{2.20}$$

For each individual sampling of an element of the training set we have

$$\mathcal{D}(\boldsymbol{x}_i : h(\boldsymbol{x}_i) = \boldsymbol{y}_i) = 1 - L_{(\mathcal{D})}(h) \leq 1 - \epsilon, \tag{2.21}$$

where the last inequality follows from the fact that $h \in \mathcal{H}_B$. Combining the previous equation with Equation (2.20) and using the inequality[4] $1 - \epsilon \leq e^{-\epsilon}$ we obtain that for every $h \in \mathcal{H}_B$,

$$\mathcal{D}^m(\{S|x : L_S(h) = 0\}) \leq (1 - \epsilon)^m \leq e^{-\epsilon m} \tag{2.22}$$

Combining this equation with Equation (2.19) we conclude that

$$\mathcal{D}^m(\{S|x : L_{\mathcal{D}}(h_S) > \epsilon\}) \leq |\mathcal{H}_B|e^{-\epsilon m} \leq |\mathcal{H}|e^{-\epsilon m}. \tag{2.23}$$

We successfully bounded the probability of a learner to fail with this inequality. As the intuition says, the bound is decreasing (and therefore giving more theoretical guarantees) as both m ϵ increase. The former suggests to increase the sample size to avoid overfitting, while the latter is an example of how slackening constraints are easier to meet. We are interested to see under which conditions the Equation (2.23) upper bounds the probability to fail by a (small) positive constant δ, in particular we want to focus on the minimal amount of datapoints that guarantees a probability of failure of at most δ. This can be achieved by setting the right hand side of the Equation to δ and isolating the m term in the inequality. The result is the following Theorem:

Theorem 2.3.1. *Let $\mathcal{H}$ be a finite hypothesis class. Let $\delta \in (0, 1)$, $\epsilon > 0$, and let m be an integer that satisfies $m \geq \epsilon^{-1} \log(|\mathcal{H}|/\delta)$.*

Then, for any labelling function f, and for any distribution $\mathcal{D}$ for which the realisability assumption holds, with probability of at least $1 - \delta$ over the choice of an i.i.d. sample S of size m, we have that for every ERM hypothesis h_S it holds that

$$L_{\mathcal{D}}(h_S) \leq \epsilon. \tag{2.24}$$

Proof. Simply inverting the relationship and isolating the m term, the proof follows. □

The preceding Theorem tells us that for a sufficiently large m, the $ERM_{\mathcal{H}}$ rule over a finite hypothesis class will be probably (with probability at least $1 - \delta$) approximately (up to an error of ϵ) correct. This leads to the definition of *sample complexity* of a hypothesis class:

Definition (Sample Complexity)**.** The *sample complexity* of a hypothesis class $\mathcal{H}$ is defined as the minimal function $m_{\mathcal{H}} : (0, 1)^2 \to \mathbb{N}$ such that for any $\epsilon, \delta,$, $m_{\mathcal{H}}(\epsilon, \delta)$ is the minimal integer that satisfies the requirements of PAC learning with accuracy ϵ and confidence δ. That is, it determines how many examples are required to guarantee a probably approximately correct solution with the given parameters.

Remark. We observe that the lower bound is most sensible to the value of ϵ, and increases logarithmically as δ and $\mathcal{H}$ increase. As hypothesis classes usually increase exponentially in size with the number of parameters, the relationship between $m_{\mathcal{H}}$ and the (parameters of) $\mathcal{H}$ is *de facto* linear. We conclude that the parameter with which we can use more flexibility is δ.

[4] valid since ϵ is ≤ 1

We conclude that that for a finite hypothesis class, if the ERM rule with respect to that class is applied on a sufficiently large training sample (whose size is independent of the underlying distribution or labelling function) then the output hypothesis will be probably approximately correct. We summarise this by the following Definition:

Definition (PAC Learnability)**.** A hypothesis class $\mathcal{H}$ is *probably almost correct (PAC) learnable* if there exist a function $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ and a learning algorithm A with the following property: For every $\epsilon, \delta \in (0,1)$, and for every distribution $\mathcal{D}$ over $\mathcal{X}$, and for every labelling function $f : \mathcal{X} \to \mathcal{Y}$, if the realisable assumption holds then when running the learning algorithm on $m \geq m_{\mathcal{H}}(\epsilon, \delta)$ i.i.d. examples the algorithm returns a hypothesis h such that, with probability of at least $1 - \delta$ (over the choice of the examples)

$$L_{\mathcal{D}}(h) \leq \epsilon. \tag{2.25}$$

Finally, combining Theorem 2.3.1 with the new definition we have the following Corollary:

Corollary 2.3.1. *Every finite hypothesis class $\mathcal{H}$ is PAC learnable with sample complexity*

$$m_{\mathcal{H}}(\epsilon, \delta) \leq \lceil\ \epsilon^{-1} \log(|\mathcal{H}|/\delta)\ \rceil \tag{2.26}$$

Which means that an algorithm will probably almost correctly learn a good model for the data with an amount of datapoints at most equal to the right hand side of Equation (2.7),

The next step is to drop the realisability assumption. When doing so a new learnability definition arises and we refer to it as *agnostic PAC learnability.*

2.3.3 Agnostic PAC learnability

Recall that the realisability assumption in page 18 requires that there exists a $h^* \in \mathcal{H}$ such that the $L_{\mathcal{D}}(h^*) = 0$. This implies that $\mathbb{P}_{\mathcal{D}}[h^*(x) = f(x)] = 1$, that is, that the labelling is almost surely correct for all samples. In many practical problems this assumption does not hold.

Moreover we shall not assume that the labels are fully determined by the features we measure on input elements. We therefore drop the such assumption and aim for a more realistic model for the data-generating distribution. Formally from now on, we will call $\mathcal{D}$ the probability distribution over $\mathcal{X} \times \mathcal{Y}$; that is, $\mathcal{D}$ is a joint distribution over both domain points and labels. Shalev-Shwartz and S. Ben-David (2014) suggest viewing such a distribution as being composed of two parts: a distribution $\mathcal{D}_x$ over unlabelled domain points (sometimes called the marginal distribution) and a conditional probability over labels for each domain point, $\mathcal{D}((x,y)|x)$. In this framework a learning algorithm does not assign a label to a datapoint in a deterministic fashion, but rather assigns a probability distribution of labels. We call such models *probabilistic models.*

Now that the realisability assumption is dropped, a minimal (non-negative) possible error $L_{\mathcal{D}.f}(h)$ exists, and we cannot hope that the learning algorithm will find a hypothesis whose error is smaller than the minimal possible error. Instead, we require that the learning algorithm will find a predictor whose error is not much larger than the best possible error of a predictor in the hypothesis

class. When the realisability assumption is dropped we talk about *agnostic learning*[5], and the the definition of *agnostic PAC Learnability* follows:

Definition 2.3.1 (Agnostic PAC Learnability)**.** A hypothesis class $\mathcal{H}$ is *agnostic PAC learnable* if there exist a function $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ and a learning algorithm A with the following property: For every $\epsilon, \delta \in (0,1)$ and for every distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, when running the learning algorithm on $m \geq m_{\mathcal{H}}(\epsilon, \delta)$ i.i.d. examples generated by $\mathcal{D}$, the algorithm returns a hypothesis h such that, with probability of at least $1 - \delta$:

$$L_{\mathcal{D}}(h) \leq \min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h') + \epsilon. \tag{2.27}$$

It is common practice to define PAC learnability with respect to a given loss function ℓ. A hypothesis class is PAC learnable wit respect to a loss function ℓ when Equation (2.27) holds, with $L_{\mathcal{D}}(h)$ defined as $\mathbb{E}_{\mathcal{D}}[\ell(h, z)]$. and $L_D(h')$ accordingly is $\mathbb{E}_{\mathcal{D}}[\ell(h', z)]$.

Remark. When the realisability assumption holds we have

$$\min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h') = L_{\mathcal{D}}(h^*) = 0$$

for some $h^* \in \mathcal{H}$ and agnostic PAC learning provides the same guarantee as PAC learning. In this sense, agnostic PAC learning generalises the definition of PAC learning. When the realisability assumption does not hold, no learner can guarantee an arbitrarily small error but can declare success if its error is not much larger than the best error achievable by a predictor from the class $\mathcal{H}$.

2.3.4 Comparison with real-world example

With theoretical bounds being set, we can compare them with the results from Equation (2.8) and Figure 2.2. The example, despite being simple, has the advantage of being easily controllable, and allows us to verify if the lower bounds for $m_{\mathcal{H}}$ as stated in Corollary 2.3.1 are consistent with our findings In order to do this, we need set the parameters δ and ϵ and calculate the remaining values. The parameter δ is the least influential one and it is common practice to set it equal to 0.05. ϵ can be fixed to 0.015, a choice we will see being consistent with the benchmark set in the Example. It is worth noting that setting $\epsilon = 0.015$ we are guaranteeing a performance at most 0.015 worse than the *best* available learner in the hypothesis class, by Definition 2.3.1. Finally, we can easily verify that $\mathcal{H}_1, \mathcal{H}_4, \mathcal{H}_{10}$, have cardinality $2^{64 \cdot 2}, 2^{64 \cdot 5}$ and $2^{64 \cdot 11}$ respectively (see page 17). With such values, we are guaranteed PAC learnability of our regression problem for the values of $m_{\mathcal{H}_k}$ shown in Table 2.1. The values are the *lower bounds* guaranteeing agnostic PAC learnability, or alternatively they *upper bound* the sample size which do not guarantees that. As we can see, the theoretical bounds are much higher than the actual number of points we used to build the model. However we can verify whether the model has "learned" already from the given 25 datapoints by comparing its performance with the bound given by (2.27).

[5]Note that the term "agnostic" will have a different meaning in Section 2.7

Hyp. Class	class size	$m_{\mathcal{H}}$ bound
$\mathcal{H}_1$	$2^{64\cdot 2}$	$m_{\mathcal{H}_1}(.015, .05) = 5716$
$\mathcal{H}_4$	$2^{64\cdot 5}$	$m_{\mathcal{H}_4}(.015, .05) = 14588$
$\mathcal{H}_{10}$	$2^{64\cdot 11}$	$m_{\mathcal{H}_{10}}(.015, .05) = 32333$

TABLE 2.1: Sample size guaranteeing agnostic PAC learnability for the three hypothesis classes $\mathcal{H}_1, HHH_4, HHH_{10}$.

- The cases $\mathcal{H}_4$ and $\mathcal{H}_{10}$ are straightforward: the "true" underlying function f is included in the sets (up to machine-precision related representation errors, which are negligible), therefore $\min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h') = L_{\mathcal{D}}(f) = 0.01$ as proved in (2.9). From Equation (2.27) we have that the model has learned a successful h when:

$$L_{\mathcal{D}}(h) \leq \min_{h' \in \mathcal{H}, \mathcal{H}_{10}} L_{\mathcal{D}}(h') + \epsilon = 0.01 + 0.015 = 0.025 \tag{2.28}$$

  Recovering the lower benchmark performance of 0.975. From Figure 2.3 we can see how the model has learned at its best when we set $\mathcal{H} = \mathcal{H}_4$ but not when $\mathcal{H} = \mathcal{H}_{10}$ because of overfitting.

- For $\mathcal{H}_1$ more calculations are needed. Finding the loss incurred by the best linear corresponds to the following:

$$\begin{aligned}
\min_{h' \in \mathcal{H}_1} L_{\mathcal{D}}(h') &= \min_{a_0, a_1 \in \mathbb{R}} \mathbb{E}\left[\ell(f(x) + \mathcal{E}, (a_0 + a_1\, x))\right] = \qquad (2.29)\\
&= \min_{a_0, a_1 \in \mathbb{R}} \mathbb{E}\left[(f(x) + \mathcal{E} - (a_0 + a_1\, x))^2\right] = \\
&= \min_{a_0, a_1 \in \mathbb{R}} \mathbb{E}\left[(f(x) - (a_0 + a_1\, x))^2 + \mathcal{E}^2 + 2\mathcal{E}(f(x) - (a_0 + a_1\, x))\right] = \\
&= \min_{a_0, a_1 \in \mathbb{R}} \mathbb{E}\left[(f(x) - (a_0 + a_1\, x))^2\right] + \mathbb{E}[\mathcal{E}^2] + 2\mathbb{E}[\mathcal{E}] \cdot \mathbb{E}[f(x) - (a_0 + a_1\, x)] \\
&= \min_{a_0, a_1 \in \mathbb{R}} \int_0^1 [x^3 + 3x^2 - 4x + 3 - ax - b]^2 dx + 0.01 + 0 \\
&= \qquad 79/700 + 0.01 \approx 0.123. \qquad (2.30)
\end{aligned}$$

  Plugging this value to Equation (2.27) leads to a successful (PAC agnostic) learning when:

$$L_{\mathcal{D}}(h) \leq 0.123 + 0.015 = 0.138. \tag{2.31}$$

  That is, when the performance is above $1 - 0.138 = 0.862$. Comparing this benchmark with the performance obtained in Figure 2.3 (around 0.88), we conclude that the model has learned successfully, but being the class is too small for our purposes we are incurring in underfitting.

The results from the former experiment are non contradicting with the bounds set by Equation (2.27), these bound are indeed *lower* bounds *guaranteeing* PAC learnability, and they are way higher than the number of datapoints we are using. Nevertheless the fitting algorithm worked wuth the hypothesis classes $\mathcal{H}_1$ and $\mathcal{H}_4$.

An explanation to this comes from the fact that the given model is extremely simple: the data is 1-dimensional and the labelling function is a (continuous) polynomial: no surprise that less points were needed to obtain results. The theoretical bounds make indeed no assumption on the complexity of the underlying model, and are valid *for any* labelling function $f : \mathcal{X} \to \mathcal{Y}$, this is normally a strong point for theoretical models but in this case it appear more like a limitation. Another fundamental weakness of such bounds lies on the chain of inequalities between Equation (2.20) and Equation (2.23) used to prove them, specifically in Equation (2.23) it is possible that $|\mathcal{H}_B| << |\mathcal{H}|$.

We conclude that the theoretical bounds from Shalev-Shwartz and S. Ben-David (2014) are non contradicting our findings, but have limitations in real world applications. We therefore need to introduce and develop other techniques to spot overfitting, rather than trying to predict the probability of this happening.

2.4 Spotting under and overfitting

In the previous Sections, we discovered how overfitting looks like and showed the most immediate technique to spot and prevent it. We also showed the theoretical proof that, the larger the cardinality of the class of learners $\mathcal{H}$, the more samples are needed for effective learning. In this Section, we will introduce more advanced techniques. The first of them is the so called *K-fold cross-validation*.

2.4.1 K-fold cross-validation

Cross-validation can be seen as a repeated train-test data split. The data is first shuffled randomly and split into K groups. The procedure goes then as the following:

- For each of the K groups, hold out one group as a test set, and train the algorithm on the other $K - 1$.
- Evaluate the performance of the trained algorithm on the test data. Retain the score after the model is discarded for the next iteration.
- Iterate the previous steps K times until all sub-group have been used once as a test set and $K - 1$ times as a training set. It is important to understand that each observation in the data is assigned to an individual group and stays in that group for the duration of the procedure.

The final output consists of K evaluation metrics based on the K generated test sets. The mean and the standard deviation of such measurements give an idea of the performance of the machine learning algorithm and of the confidence interval of such estimate. The procedure can be visualised in Figure 2.4. The value of K is a parameter whose value is up to the user. The choice of a

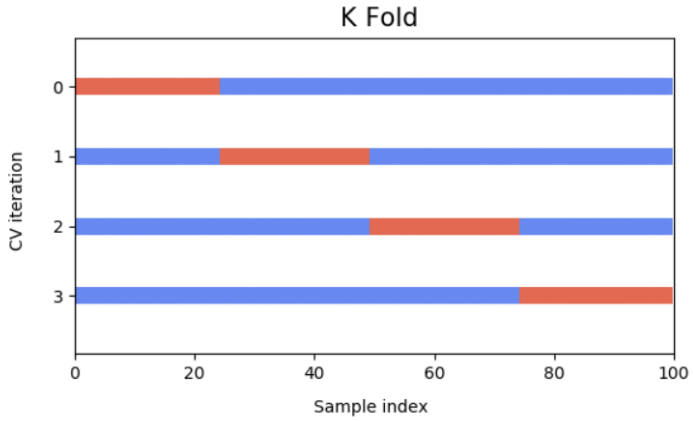

FIGURE 2.4: Example where a 4-fold cross validation is performed. Approximately 25 % of the data is left out during each training phase and used to evaluate the performance. Picture adapted from https://scikit-learn.org/stable/modules/cross_validation.html

small K creates a higher selection bias on the performance metric but low variance. A large K induces little bias but increases the variance on the measured performances. A typical value for K is 10, which results in ten $90\% - 10\%$ splits during the procedure. Cross validation not only has the advantage of giving a clear overview of the algorithm's performance through the average and standard deviation of the metrics, but also guarantees better robustness to outliers than a simple train-test split. Unlike the latter, where outliers can end up being under or over-represented in the train set (and the performance metric being therefore over or under-estimated), the cross-validation procedure guarantees the averaging-out of such effects by the end of the cycle.

We conclude that K-fold cross validation makes use of the data more efficiently than a simple train-test split, and together with its variants such as *stratified K-Fold* and *group K-Fold* is a state of art technique when it comes to performance assessment in scientific publications.

2.4.2 ABN validation procedure

Given the big societal responsibilities that a bank like ABN Amro has, the performance assessment of its own models and tools, including machine learning models, needs to comply to the highest standards. It is extremely important to avoid overfitting and performance overestimation, and the reasons are the same mentioned in the importance of explainability in the Introduction.

In this Section we describe how ABN Amro manages the dataset and what techniques are used to prevent overfitting. The details of the methods might be sensitive information and are commented out.

2.5 Machine Learning algorithms

We mentioned Machine Learning algorithms throughout this work and we described them as algorithms who improve their guesses based on the data they receive. A well-tuned machine learning algorithm will get better as more data is fed in and the resulting process reminds what humans would call "learning". The process of learning in ML is inherently different from how humans and animals learn, but the word has entered the community's vocabulary and gives a good intuition. It is a common (see Molnar (2019) and Murphy (2012)) to split the field of ML in three sub-fields. Namely we talk about *supervised learning*, *unsupervised learning* and *reinforcement learning*.

Supervised learning algorithms build models based on a set of data where both $\mathcal{X}$ and $\mathcal{Y}$ are non-empty. The algorithm predicts the output of new data and its performance is measured through a penalty function. The algorithm aims at minimising the loss given by such function while making its guesses, and to do so it goes first through the so called "training process" with the known data. Examples of supervised learning algorithms include regression and classification algorithms. The first type is used when the predicted output lies within a numerical range in the continuum, the second one when the outputs are restricted to a limited set of values, numerical or not. When not all output values are available in the data set, and the goal is to predict (usually, to classify) the data with no y-value, in these cases we talk about *semi-supervised learning*. Figure 2.5 shows an example of supervised learning, where the algorithm learns to classify emails into "spam" and "not spam".

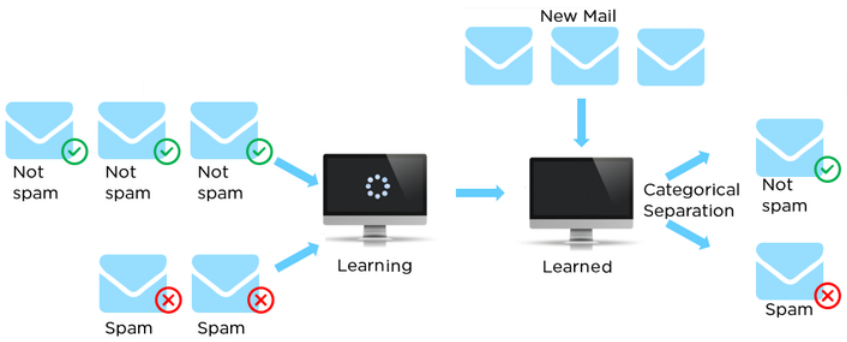

FIGURE 2.5: Visualisation of the supervised learning process: 1) Labelled data is given to the machine. 2) The algorithm learns patterns from the data and elaborates its own classification rules. 3) These rules are applied to new unlabelled data and the classification is done. Adaptation of https://towardsdatascience.com/what-are-the-types-of-machine-learning-e2b9e5d1756f

The second category consists of unsupervised learning algorithms. These algorithms are used when no output data is available, that is when $\mathcal{Y} = \emptyset$ and the challenge is to find structure in the data even if no distinction between training and test data is possible. A common setup consist in a input set $\mathcal{X}$, and a distance or similarity function over it. The algorithm then returns a partition of the domain set $\mathcal{X}$ into subsets $(C_1, \ldots, C_k)$ called clusters. In other words, clustering algorithms assign a label to each data point so that data points falling in the same cluster are similar according to some criteria, while data points in different clusters are not.

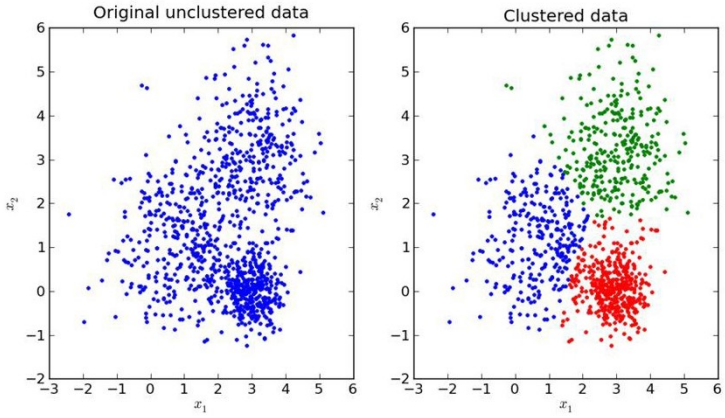

FIGURE 2.6: In unsupervised learning the original data has no labels. A K-Means algorithm is separating the data in three clusters by proximity. Example taken from http://pypr.sourceforge.net/kmeans.html#k-means-example

Finally, there are the so called reinforcement Learning algorithms. According to Shalev-Shwartz and S. Ben-David (2014), this kind of learning can be seen as an intermediate setting where training examples contain more information than the test examples. The learner, usually called "agent", is required to predict even more information for the test examples, and it does so in an interactive environment by trial and error using feedback from its own actions and experiences. In reinforcement learning the goal is to find a suitable action model that would maximise the total cumulative reward of the agent.

In this work, we focus on the supervised learning algorithms. Both applications shown in Chapter 3 will involve an algorithm for supervised learning called *Stochastic Gradient Boosting* (SGB). We link this algorithm to the previously introduced theoretical framework of Sections 2.3.2 - 2.3.3, and we do this by introducing the concept of *weak learning*, and linking this to a ML algorithm called *Adaptive Boosting* (AdaBoost) *algorithm*.

2.5.1 Weak Learning

The concept of *weak learnability* arises as a contrast to the concept of (strong) PAC learnability introduced in page 21. We saw that strong learners are able to find an arbitrarily good classifier: for any given ϵ they can learn a function whose error is $\leq \epsilon$. Weak learners on the other side are classifiers that are only slightly correlated with the true classification, that is they can label examples slightly better than random guessing. How "slightly" better than random guessing should the labelling be is regulated by the parameter γ. Shalev-Shwartz and S. Ben-David (2014) chooses to define weak learnability in case of a binary classification first, but the notion is easily extendable to multi-class and regression problems.

Definition (Weak Learner)**.** A learning algorithm A is called a γ-weak learner for a class $\mathcal{H}$ if there exists a function $m_{\mathcal{H}} : (0,1) \to \mathbb{N}$ such that for every $\delta \in (0,1)$, for every distribution $\mathcal{D}$ over $\mathcal{X}$, and for every labelling function $f : \mathcal{X} \to \{\pm 1\}$, if the realisable assumption holds with respect to $\mathcal{H}, \mathcal{D}$ and f, then when running the learning algorithm on $m \geq m_{\mathcal{H}}(\delta)$ i.i.d. examples generated by $\mathcal{D}$ and labelled by f, the algorithm returns a hypothesis h such that, with probability of at least $1 - \delta$:

$$L_{\mathcal{D}}(h) \leq \frac{1}{2} - \gamma. \tag{2.32}$$

A hypothesis class $\mathcal{H}$ is called γ-weak-learnable if there exists a γ weak learner for that class.

At this point we might ask ourselves what is the point of introducing weak learners since strong ones are already available. Moreover, strong and weak-learners are equivalent from the statistical perspective: a hypothesis class $\mathcal{H}$ is strong learnable if and only if is weak-learnable (Shalev-Shwartz and S. Ben-David, 2014, p.103). In a purely theoretical framework these remarks are true, however there are two main (practical) drawbacks of strong learning that we wish to overcome:

- Firstly, implementing an algorithm to search for the best strong learner using the $\text{ERM}_{\mathcal{H}}$ rule is computationally expensive (NP-hard according to Ben-David, Eiron, Long, et al. (2003)).
- The theoretical framework in of Sections 2.3.2 - 2.3.3 has not been applicable to our work so far.

We are now motivated in the choice of using weak-learners, and we wish to replicate the performance guaranteed by strong learners. The AdaBoost algorithm successfully implements a combination of weak learners to obtain a performance equivalent to a strong learner. Informally speaking, the principle behind this is that "the performance of an average of weak learners is better than the average of the performance of the single weak learners".

2.5.2 AdaBoost algorithm

The AdaBoost algorithm was first proposed Schapire and Freund in 1995 (Shalev-Shwartz and S. Ben-David, 2014). This algorithm is the first truly practical implementation of boosting, that is of the concept that efficient weak learners can be combined to form a strong one. AdaBoost became

hugely popular and Freund and Schapire's work has been recognised by prestigious awards such as the Gödel Prize. The AdaBoost algorithm can be started by receiving as input:

- A training set of examples $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$ where a labelling function f such that $f(\boldsymbol{x}_i) = y_i \ \forall i \leq m$ is assumed to exists[6].
- A distribution vector over the samples S denoted by $\boldsymbol{D}^{(1)} = (D_1^{(t)}, \ldots, D_m^{(t)})$. Usually $\boldsymbol{D}^{(1)}$ is the uniform distribution over the m samples.
- Lastly, a maximum number rounds T (or alternatively a stopping criterion) is given.

The boosting proceeds in a sequence of consecutive rounds, and for each round $t \leq T$:

- A weak learner returns a "weak hypothesis" h_t. This learner incurs in an error:

$$\epsilon_t = L_{\boldsymbol{D}^{(t)}}(h_t) := \sum_{i=1}^{m} D_i^{(t)} \mathbf{1}_{[h_t(\boldsymbol{x}_i) \neq y_i]}, \tag{2.33}$$

  which by definition is at most $\frac{1}{2} - \gamma$ with probability at least $(1 - \delta)$.

- AdaBoost assigns a weight for the learner h_t according to the formula:

$$w_t(\epsilon_t) = \frac{1}{2} \log\left(\frac{1}{\epsilon_t} - 1\right). \tag{2.34}$$

  This weight is decreasing with respect to the error.

- Finally the distribution $\boldsymbol{D}^{(t)}$ is updated and normalised to $\boldsymbol{D}^{(t+1)}$ for the next step. Componentwise:

$$D_i^{(t+1)} = \frac{D_i^{(t)} \exp(-w_t \, y_i \, h_t(\boldsymbol{x}_i))}{\sum_{j=1}^{m} D_j^{(t)} \exp(-w_t \, y_j \, h_t(\boldsymbol{x}_j))}, \quad \forall i \leq m. \tag{2.35}$$

  This operation is called *exponential weighting*, and it is forcing the weak learner to focus on the problematic examples in the next round. As a result, by adding more weight to the misclassified samples, a weak learner is forced to perform much better in the examples with bigger weight $\boldsymbol{D}^{(t)}$ in order to incur in a loss smaller than $\frac{1}{2} - \gamma$.

After T rounds, the AdaBoost algorithm for binary classification outputs the hypothesis:

$$\hat{f}_S(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{t=1}^{T} w_t h_t(\boldsymbol{x})\right). \tag{2.36}$$

[6] In our deterministic binary classification example, f exists indeed.

Remark. Each weak learner e_t incurs in an error $\leq \frac{1}{2} - \gamma$ with probability $(1 - \delta)$. This means that the probability that *every* weak learner does so is:

$$(1 - \delta)^T > (1 - \delta T). \tag{2.37}$$

Meaning that, on the other side, the probability of the algorithm failing at some round is upper bounded by δT. As we have seen in the Remark of page 20 that we can easily play with the δ parameter.

From ADABoost to Gradient Boosting

As we have seen, AdaBoost changes the sample distribution by modifying the weights attached to each of the instances at each iteration.The weights of the wrongly predicted instances are increased and the ones of the correctly predicted instances are decreased. The weak learner thus focuses more on the difficult instances and the higher it performs, the more it contributes to the strong learner. An explanatory image to how the error ϵ_t influences the weight is shown in Figure 2.7.

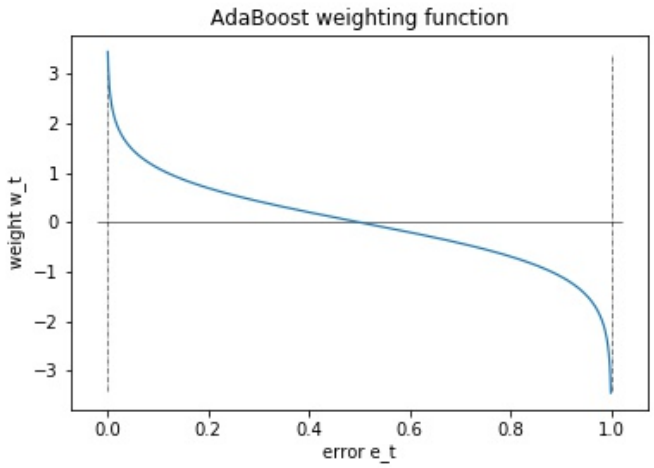

FIGURE 2.7: Weighting function as a function of the error ϵ_t, good learners get a positive weight, tending to infinity as the error approaches to zero. Symmetrically, bad learners have a negative weight, and $w_t(0.5) = 0$.

An alternative approach come from Gradient Boosting. Gradient Boosting is another supervised ML algorithm which has many traits in common to AdaBoost. Gradient Boosting does not modify the sample (weights of the) distribution but trains the weak learner on the remaining errors (so-called pseudo-residuals) instead. This is another way to give more importance to the difficult

instances. In the next Section we see in details how Gradient Boosting and its Stochastic variation works.

2.5.3 (Stochastic) Gradient Boosting

The Gradient Boosting algorithm was first proposed by Friedman in 1999 and it is used in supervised learning problems. To describe how the algorithm works we refer to the 2001 version of the paper (Friedman, 2001).

The supervised learning setting is the usual one: we are given a training set S consisting of m datapoints $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$ and a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. Our aim is to find an approximation function $\hat{f}(\boldsymbol{x})$ to the model such that it minimises the expected loss:

$$\hat{f} = \arg\min_{f} \mathbb{E}[\ell(y, f(\boldsymbol{x}))]. \tag{2.38}$$

Like other boosting methods, Gradient Boosting does so by combining weak learners $h_i \in \mathcal{H}$ into a strong one in an iterative fashion. The algorithm seeks such approximation in the form of weighted sums of functions, a principle we have seen in AdaBoost. At every step $t \leq T$ a new weak learner is added to the previous learned function. Running a Gradient Boosting algorithm after T steps leads to the predictor:

$$\hat{f}_T(x) = \sum_{t=1}^{T} \gamma_t h_t(\boldsymbol{x}). \tag{2.39}$$

Where h_t stands for the learner added at the t-th iteration step, and γ_t refers to the weight associated to it. We are not using the same notation as AdaBoost for the weights because the construction in Gradient Descent is inherently different. For the first step ($t = 0$), Gradient Boosting searches for the best possible constant predictor:

$$\hat{f}_0(x) = \arg\min_{\gamma} \sum_{i=1}^{m} \ell(y_i, \gamma), \tag{2.40}$$

Then, for $1 \leq t \leq T$ the method the algorithms applies a steepest descent step to this minimisation problem. The resulting step is described by the following equations:

$$\hat{f}_t(x) = \hat{f}_{t-1}(x) - \gamma_t \sum_{i=1}^{m} \nabla \ell(y_i, \hat{f}_{t-1}(x_i)), \tag{2.41}$$

$$\gamma_t = \arg\min_{\gamma} \sum_{i=1}^{m} \ell\left(y_i, \hat{f}_{t-1}(x_i) - \gamma \nabla \ell(y_i, \hat{f}_{t-1}(x_i))\right). \tag{2.42}$$

Where the nabla (∇) operators stand for the derivatives with respect to the second argument of ℓ, that are the functions $\hat{f}_t, i \leq m$. It is worth noting that such derivatives may not be elements of

$\mathcal{H}$. In such cases, the closest candidate function h_t to the gradient of ℓ is chosen. The coefficient γ_t is heuristically found by solving a 1-dimensional optimisation problem.

Remark. This approach is a heuristic one for two reasons. Firstly, Equation (2.41) gives an approximated value to the greedy increment from step $t-1$ to step t. Such step would ideally be:

$$\hat{f}_t(x) = \hat{f}_{t-1}(x) + \underset{h_t \in \mathcal{H}}{\arg\min} \left[\sum_{i=1}^{m} \ell(y_i, \hat{f}_{t-1}(x_i) + h_t(x_i)) \right]. \tag{2.43}$$

Unfortunately a searching algorithm within a possibly large $\mathcal{H}$ makes this step computationally expensive, especially if it needs to be done at every step t of the algorithm. Secondly, only an approximated value for the γ_t coefficient on the optimisation problem is found at every step, as finding the exact solution to equation would be too expensive.

Remark. In a later paper by Hastie, Tibishirani, and Friedman (2009) a regularising shrinking parameter ν is suggested for the gradient boosting method. This modifies Equation (2.41) into the following:

$$\hat{f}_t(x) = \hat{f}_{t-1}(x) - \nu\,\gamma_t \sum_{i=1}^{m} \nabla \ell(y_i, \hat{f}_{t-1}(x_i)),. \tag{2.44}$$

With $\nu < 0.1$ a better generalisation power is reported, alhough it increases computational time (a larger T is needed).

We now show how Gradient Boosting works with a simple example taken from Grover (2017). The underlying process to be learned is a function $f : \mathbb{N} \to \mathbb{R}$ defined as the following:

$$\begin{cases} Y \sim \text{Unif}(10, 15) & \text{for } \ 0 \leq X < 10 \\ Y \sim \text{Unif}(20, 25) & \text{for } 10 \leq X < 20 \\ Y \sim \text{Unif}\ (\ 0,\ 5) & \text{for } 20 \leq X < 30 \\ Y \sim \text{Unif}(30, 32) & \text{for } 30 \leq X < 40 \\ Y \sim \text{Unif}(13, 17) & \text{for } 40 \leq X < 50 \end{cases} \tag{2.45}$$

The weak learners h_t are picked from the class of decision trees, and the L_2 loss is being used. The behaviour of the Gradient Boosting algorithm under this setting is shown in a controlled step-by-step fashion in Figure 2.8.

The top left picture shows how the constant predictor $\hat{f}_0(x) \approx 17$ is being improved with the learner h_1. The learner in this case behaves similarly to a step function which, given the one-dimensional nature of the data, is not a surprise.

Two rounds later we reach the situation in the top right picture. Three weak learners have been added to the initial guess and the algorithm is capturing some trend in the data. In particular, the datapoints indexed between 0 and 9, and between 40 and 49 are being predicted with a satisfactory accuracy. Outside these intervals the performance is still low and presents underfitting.

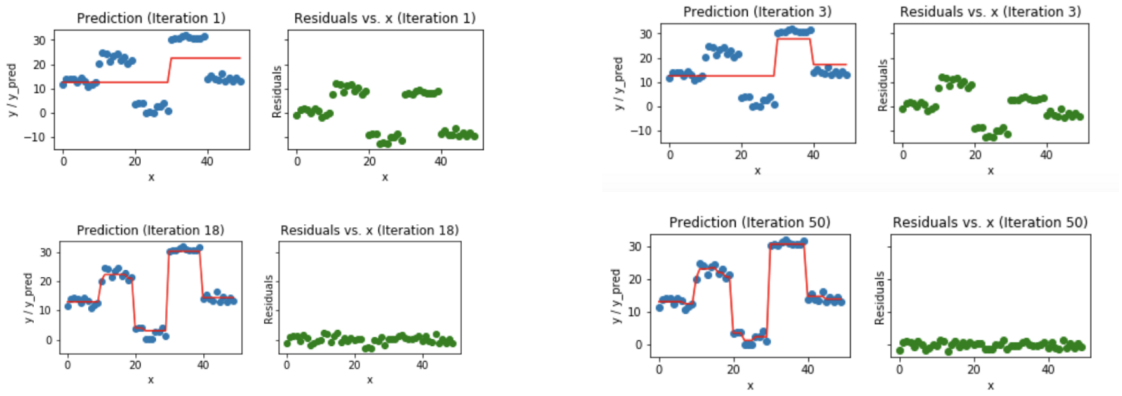

FIGURE 2.8: Gradient Boosting at work, where rounds $t = 1, 3, 18, 50$ are shown. The values of the residuals are plotted on the right. Credits to Grover (2017) for the Figure

The bottom left picture shows how $\hat{f}_{18}$ looks like. The algorithm is capturing the structure of the data: the five different clusters of data have been identified and the prediction is close to the actual values. There is limited wobbling behaviour, showing that the algorithm is not fitting the noise yet and the plot of the residuals shows that they are well distributed around zero (small bias). We conclude that the model has a good fit at this stage.

Finally, the bottom right picture shows what this Gradient Boosting algorithm does when learning for 50 rounds. The predictor follows the data well, but some wobbling appears within the clusters of data. This suggests that $\hat{f}_{50}$ is capturing and modelling some noise as well, leading to overfitting.

This example shows that setting T too high can lead to overfitting. One way to avoid this is to insert an alternative stopping criterion, for example one can tell the Gradient Boosting to stop when the performance increase on the train test increases by less than a small margin, or when the empirical risk evaluated on the test set starts to rise. However, there also exists a variant of Gradient Boosting which is more robust to overfitting. The same caveats apply, but the overfitting is expected to be less severe.

Stochastic Gradient Boosting

The stochastic variant of the algorithm was introduced by the same Friedman (Friedman, 2001) and it consists in a very simple concept. At each iteration of the algorithm, the base learner candidates fit from a sub-sample of the training set instead of the whole one, with 50% to 80% of the datapoints being sampled at every iteration. This way, the accuracy of the algorithm is reported to improve, variance to be reduced and overfitting to be somewhat prevented. Even computational time is decreased as at each step as the fitting is done on a smaller sample.

Thanks to its robustness and effectiveness Stochastic Gradient Boosting (SGB) has become a very popular ML algorithm and we will see two applications of it in Chapter 3. On the other side, the outputs of the SGB algorithm an be hard to explain especially with complex datasets. Already in the previous, simple example, some questions are left unanswered: why are exactly learners $\{h_1, h_2, \dots, h_{50}\}$ picked? When does the model start overfitting, precisely? How do we define a stopping criterion over another to prevent overfitting? It is clear that, although a well-tuned SGB algorithm performs well, it is not well explainable in the current state.

2.6 Interpetability of models

In the previous Sections we focused on the theoretical background of some ML algorithms and we introduced valuable techniques to assess their performance. However, the Introduction of this work highlighted that in certain contexts *interpretability* (or *explainability*) alongside performance is determining whether a ML algorithm can be implemented. From now on, we should therefore focus on this other aspect, define what interpretability is and introduce techniques whose aim to make hardly-interpretable model more approachable for their end-users. We start by giving a definition of interpretability:

Definition (Interpretability)**.** Interpretability of a ML algorithm is the degree to which the user can understand the cause of a decision and consistently predict the algorithm's result.

The reader might notice that this definition is subjective: there is no mathematical measure for interpretability, but it rather depends on the background of the model's user. We will therefore limit ourselves into classifying ML algorithms into "high", "medium" and "low" interpretability, assuming the user has a quantitative background on the field but can not be called an *expert* of the field, just like the writer of this Thesis. We end up with the following classification:

- ML algorithms such as Logistic Regression, Linear Regression, Decision Trees are considered to be highly interpretable.
- Algorithm such as K-Nearest Neighbours and Random Forest lie somewhere in the middle of the spectrum.
- Methods involving (Stochastic) Gradient Boosting, Support Vector Machines (SVM) and Neural Networks (NN) are usually hard to explain.

In the next Section we investigate whether there is a connection between interpretability and predictive power of an algorithm. We will see that a trade-off arises when balancing these two features. That is, an increase in predictive power usually comes at a cost of worse interpretability; vice-versa, highly interpretable models tend to be outperformed by less-explinabe counterparts.

2.6.1 The Power vs Interpretability trade-off

Intuition suggests that more complex models can be more accurate, but are less understandable and controllable. This concept can be applied in any ML algorithms, from simple linear regressions to Stochastic Gradient Boosting to Neural Networks.

The interpretability of a simple linear regression is straightforward when a limited amount of variables is included, but can become challenging when variables add up to thousands many. Unsurprisingly, such mega-dimensional regressions are rare, they present overfitting issues and correlated features jeopardise the explainability of the model. For this reason, we consider linear regressions with a limited amount of variables. This technique is highly explainable, but works only in a limited context where no interactions among the given variables occurs. We conclude that linear regression is a highly-explainable technique but has limited predictive power.

Similarly, Decision Trees is a highly explainable technique: finding the splitting rules is sufficient to understand the algorithm's outcome. The predictive power of Decision Trees is limited, mainly due to overfitting, but a development called Random Forest increases its predictive power at the price of some interpretability. A Random Forest method operates by constructing a multitude of Decision Trees during the training phase time and outputs the average prediction of the individual trees. We conclude that Random Forests have a "medium" interpretability and an average predictive power.

In the class of medium interpretability and average predictive power we place an unsupervised learning technique such as the K-means-clustering algorithm. The construction of the algorithm is straightforward, but the outputs can be counter intuitive in high dimensional cases. The predictive power is hard to assess ad is highly dependent on the quality of the data, we therefroe place it in the "average" side of the spectrum.

Finally, the most highly praised models such as Stochastic Gradient Boosting and Neural Networks are hard to explain. The human user cannot follow the *rationale* behind the choice of the weak learners at every step in case of SGB, nor can follow how signal is propagated in NN and how thousands of parameters are tuned in the meantime. We conclude that SGB and NN have a highly predictive power but a low interpretability. Similarly, the top performing algorithms in the Kaggle Data Science challenges result in a blend of several predictors, making them complex and lacking of interpretability. The situation is summarised in Table 2.2:

The Table shows a clear trend: the more predictive power the algorithms have, the less explainable they tend to be. This impression is not only a personal remark, but also mentioned throughout papers interested in explainable ML and AI (Rodriguez, 2018).

For this reason, efforts are being made to make the more complex model more interpretable, especially the ones appearing in the bottom right corner of the Table. There are many ways to do so, we will group them in three categories:

- **Feature selection:** consists in explaining how features are selected for a specific input. Feature selection can both be made on a global level, where the most significant features are included in the model, or at the local level. In this case we talk about *feature importance*,

High	- Lin. Regression - Decision Trees		
Medium		- K-Mean-Clusters - Random Forests	
Low			- Stochastic GB - Neural Networks
Interpretab. / Pred. Power	Low	Medium	High

TABLE 2.2: Rise of the Power vs. Interpretability trade-off: the most performing models tend to be the least explainable ones.

and consists of showing to the user which features are the most determining one for the final prediction of a specific datapoint.

- **Local Interpretability:** aims at understanding the relationship between each feature and the output on a section of the network. Local interpretability aims at building a smaller more interpretable model around the analysed datapoint(s), an ensemble of such local models can be used as a global interpretation.
- **Global Interpetability:** How well can we understand the relationship between each feature and the output value across the network. This is usually obtained by perturbing the input across the domain and observing the change of the output values.

Our choice falls in techniques focusing on global interpretability. This choice can be less effective than an ensemble of locally interpretable models, but it is less sensitive to errors and outliers. Given the high stakes and risks for a corporation like ABN Amro a more conservative approach is needed, and globally explainable techniques shall be used a this stage. In particular, we will focus our attention in two visualisation techniques, who ot only are "global" but also "agnostic", that is, model independent.

2.7 (Model agnostic) Visualisation Techniques

A *model-agnostic* interpretation consists in a set of techniques used to explain the outcome of a black-box model independently from the underlying model. As the name suggests (agnostic = form ancient Greek "no knowledge") such techniques do not use any previous knowledge about the the underlying learned model and can therefore be used under very general assumptions. As a result, these techniques can potentially also explain all those "black box" models which lie on the bottom right of the interpretability spectrum in Table 2.2. These model-agnostic interpretation methods are opposed to model-specific ones, and they are getting popular due to the increase in use of models of difficult interpretation such as Neural Networks.

In this Section, we will span through different visualisation techniques developed though the years meant for explaining black-box models. The first technique of this kind can be attributed to Friedman (2001) and is the Partial Dependency Plot.

2.7.1 Partial Dependency Plot

The first technique we show in this work is the so called Partial Dependence (PD) plot. This technique generates plots who grasp the functional relationship between a variable annd the overall output. Friedman's Partial Dependency plot (PD plot) is well explained and developed by Goldstein et al. (2015), we follow the explanation and the reasoning from his article from now on.

The technique plots the change of the average predicted value of the response of a model as some of the features vary over their marginal distribution while the others are kept fixed. More formally let $\mathcal{X} = \mathbb{R}^d$ and let $\boldsymbol{x} = (x_1, \ldots, x_d)$ be an element of the training[7] dataset. We fix the value of the inputs for a given set of indexes $S \subset \{1, ..., d\}$, and we calculate the value of the output given the inputs within the complement set C. The average response is registered as f_S. In formulas:

$$f_S(\boldsymbol{x}_S) = \mathbb{E}\left[f(\boldsymbol{x}_S, \boldsymbol{x}_C)\right] \int f(\boldsymbol{x}_S, \boldsymbol{x}_C) dP(\boldsymbol{x}_C | \boldsymbol{x}_S = x_S), \tag{2.46}$$

where $dP(\boldsymbol{x}_C|\boldsymbol{x}_S = x_S)$ indicates the marginal distribution of the $\boldsymbol{x}_C$ coordinates given the observed x_S and f_S indicates the restriction of f given fixed values $\boldsymbol{x}_S$. Usually though, such marginal distribution given observations is not known, nor is the true model f available but its ML-retrieved approximation $\hat{f}$. The success of the PD Plot relies on the following series of approximations:

$$f_S(\boldsymbol{x}_S) = \int f(\boldsymbol{x}_S, \boldsymbol{x}_C) dP(\boldsymbol{x}_C | \boldsymbol{x}_S = x_S) \approx \int f(\boldsymbol{x}_S, \boldsymbol{x}_C) dP(\boldsymbol{x}_C) \approx \tag{2.47}$$

$$\approx \int f(\boldsymbol{x}_S, \boldsymbol{x}_C) dP(\boldsymbol{x}_C) \approx \int \hat{f}(\boldsymbol{x}_S, \boldsymbol{x}_C) dP(\boldsymbol{x}_C) \approx \tag{2.48}$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} \hat{f}(\boldsymbol{x}_S, \boldsymbol{x}_{C_i}) =: \hat{f}_S. \tag{2.49}$$

Now, equation (2.47) assumes that the covariates are independently distributed, Equation (2.48) approximates the true model with the ML learned one, and (2.49) calculates the numerical discretisation of the integral. The $\boldsymbol{x}_{C_i}$ in the last line stand for the observed values of $\boldsymbol{x}_C$ in the training data. The equation on which the PD Plot techniques relies is therefore:

$$\hat{f}_S = \frac{1}{N} \sum_{i=1}^{N} \hat{f}(\boldsymbol{x}_S, \boldsymbol{x}_{C_i}), \tag{2.50}$$

[7] we follow the paper, but using the testing data also works.

And the PD Plot is created by varying the values of x_S within given intervals. Usually the interval lies between the minimum and maximum observed value of x_S. The hope is that the estimator $\hat{f}_S$ is a good approximation of the true f_S. However, this is not always the case and we will see how this technique fails to perform well in at least two situations.

Firstly PD plots can fail to spot interactions among variables of a model. An *interaction* between variables is defined to happen when the effect of these on the outcome is not additive. In real-world scenarios the change in value of a variable can trigger different outcomes, but the averaging effect of the PD plot can fail to detect such sub-trends, especially when they tend to cancel-out each other. We show an easy example where an interaction between variables is present but such information is not captured by the PD plot, irregardless of whether $\hat{f}_S$ is a good approximation of f_S or not.

Let us generate $N = 10000$ independent observations driven by the following process:

$$Y = X_1 - 5X_2 + 10X_2\mathbb{1}_{\{X_3 \geq 0\}} + \mathcal{E}, \qquad X_i \overset{iid}{\sim} \text{Unif}(-1, 1) \tag{2.51}$$
$$\mathcal{E} \overset{iid}{\sim} \mathcal{N}(0, 1/4)$$

We can see how there is an interaction between the X_2 and X_3 variable, in particular X_2 is interacting with the sign of X_3 through a product. If X_3 is positive, increasing values of X_2 have a positive effect on the response Y, while negative values for X_3 trigger a decreasing trend for Y when increasing X_2. If we run a Stochastic Gradient Boosting algorithm and build a PD Plot on the response while varying x_2 though, we get something like Figure 2.10 :

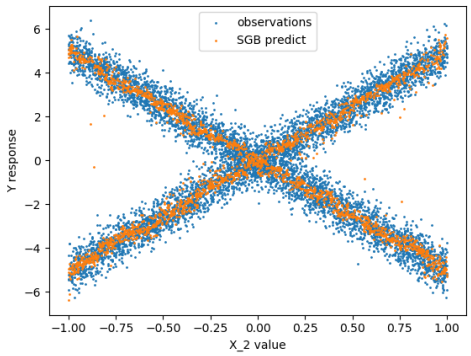

FIGURE 2.9: The SGB model is performing well.

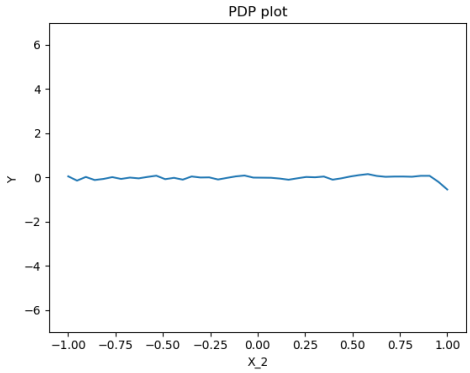

FIGURE 2.10: But the Partial Dependency Plot of x_2 is misleadingly flat.

As we can see, the response is quite flat and suggests there is no contribution given by X_2 to the prediction of the outcome Y. This is clearly a wrong conclusion as we know that X_2 has a significant contribution to the response. What is happening here is that positive and negative contributions

from X_2 tend to cancel out due to the averaging effect of PD plot. In the coming paragraphs we prove that this failure is intrinsic to the nature of the PD plot's rather than inaccuracies of the Machine Learning model. We do this by proving that even for an ideal model perfectly replicating the process in Equation 2.51, the resulting PD plot will be close to flat with a high probability.

Proof

The proof consists in showing that, given a model replicating the process in Equation (2.51), the PD Plot of the responses Y over X_2 will be flat. We do this by proving that the *average* of the Y-s is clsoe to zero for independently from the value of X_2. TO do this, we first of rewrite Y as the following:

$$Y = X_1 + 5X_2W + \mathcal{E} \tag{2.52}$$

where W is a Rademacher-distributed random variable: its outcomes are in $\{\pm 1\}$ and happen with probability $1/2$ each. Our aim is to study the distribution of the average of a sample $Y_j \sim Y,\ j \in \{1, ..., N\}$. In particular, for a given ε we want find an upper bound for:

$$\mathbb{P}\left[\left|\frac{\sum_{j=1}^{N} Y_j}{N}\right| > \varepsilon\right] \tag{2.53}$$

That is, the probability that the average over N sample of Y will vary from zero by more than ε. Our aim is to show that this probability is arbitrarily small as N increases, given any fixed ε. First of all we rewrite the content of the absolute value as the following:

$$\frac{1}{N}\sum_{j=1}^{N} Y_j = \underbrace{\frac{1}{N}\sum_{j=1}^{N} X_{1,j}}_{=A} + \underbrace{\frac{1}{N}\sum_{j=1}^{N} 5\, X_{2,j} W_j}_{=B} + \underbrace{\frac{1}{N}\sum_{j=1}^{N} \mathcal{E}_j}_{=C} \tag{2.54}$$

We now bound the probability from above:

$$\mathbb{P}\left[\left|\frac{\sum_{j=1}^{N} Y_j}{N}\right| > \varepsilon\right] = \mathbb{P}[|A + B + C| > \varepsilon] \leq \tag{2.55}$$
$$\leq \mathbb{P}[|A| > \varepsilon/3] + \mathbb{P}[|B| > \varepsilon/3] + \mathbb{P}[|C| > \varepsilon/3]$$

Where we used the triangular inequality in the second line. We now evaluate and upper bound these three contributions. For the first two ones we note that $\mathbb{E}[A] = \mathbb{E}[B] = 0$ and $\mathbb{P}[-1 \leq A \leq 1] = \mathbb{P}[-5 \leq B \leq 5] = 1$. We can therefore use Hoeffding's Lemma as stated in the Appendix to

get the following bounds:

$$\mathbb{P}[|A| > \varepsilon/3] \leq 2\exp\left(-N\,\varepsilon^2/18\right) \tag{2.56}$$

$$\mathbb{P}[|B| > \varepsilon/3] \leq 2\exp\left(-N\,\varepsilon^2/450\right) \tag{2.57}$$

The third contribution from C is a mere average over normal random variables with variance equal to $1/4$, therefore $C \sim \mathcal{N}(0, \frac{1}{4\,n})$. We cannot use the boundedness argument here anymore, but we can easily bound the probability of C being bigger than ε using a result from Theorem A.1.3 in the Appendix. We get:

$$\begin{aligned}\mathbb{P}[|C| > \varepsilon] =& 2\,\mathbb{P}[C > \varepsilon] = \; 2\,\mathbb{P}[\mathcal{N}(0, \frac{1}{4N}) > \varepsilon] = 2\,\mathbb{P}[\mathcal{N}(0,1) > 2\,\varepsilon\,\sqrt{N}] = \\ &= 2\,\Phi^c(2\,\varepsilon\,\sqrt{N}) < \frac{2}{\sqrt{2\,\pi}}\frac{1}{2\,\varepsilon\,\sqrt{N}}e^{-2\,\varepsilon^2 N} = \frac{e^{-2\,\varepsilon^2 N}}{\varepsilon\sqrt{2\,\pi N}}\end{aligned} \tag{2.58}$$

Summing up contributions from (2.56), (2.57), and (2.58) we get the following overall bound:

$$\mathbb{P}\left[\left|\frac{\sum_{j=1}^{N} Y_j}{N}\right| > \varepsilon\right] < 2\exp\left(-N\,\varepsilon^2/18\right) + 2\exp\left(-N\,\varepsilon^2/450\right) + \frac{1}{\varepsilon\sqrt{2\,\pi N}}\exp\left(-2N\,\varepsilon^2\right) \tag{2.59}$$

We set $N = 10000$ and we see how quickly does this probability converge to zero as ε increases. We do this by explicitly plotting such upper bound in Figure 2.11. Where the thin back horizontal line represents a probability of 0.05, the crossing with such line happens when $\varepsilon \approx 0.41$. That is, at least 95% of the values of the PD plot are expected to lie within the interval $[-0.41, 0.41]$. If we want to bound the probability that *all* N values of the PD plot lie within a certain ε, we can make use of the fact that points are independent in the PD plot framework. Let p be the probability of one single point falling outside the interval. The probability that all N points fall within the interval is, using independence:

$$(1-p)^N. \tag{2.60}$$

Our goal is to find p such that $(1-p)^N > 0.95$. We do this by remembering that the converging series:

$$B_n = \left(1 + \frac{\alpha}{n}\right)^n \to e^{\alpha}. \tag{2.61}$$

This series converges very fast and the value for N we are using is big enough to hold as a good approximation. This helps us observing that if we set p to $1/(20\,N)$ in Equation (2.60) we obtain:

$$\left(1 - \frac{1}{20N}\right)^N \approx e^{-\frac{1}{20}} > 0.95, \tag{2.62}$$

that is, if we bound the probability of a single pint to fall out of a given interval by $1/(20 \cdot N)$, then with probability $(1 - 1/20\,N)$ it falls withini, and with probability $> 0.95\%$ all N the points will

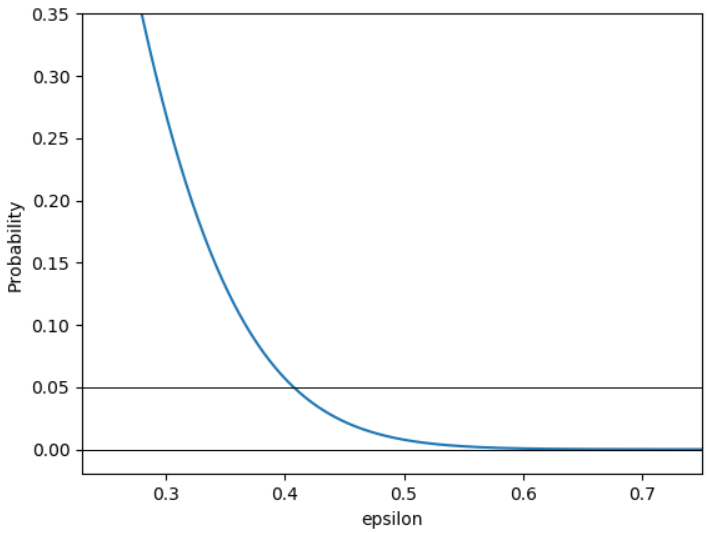

FIGURE 2.11: Upper bound of the probability of a point in the PD Plot curve to fall outside $[-\varepsilon, +\varepsilon]$, expressed as a function of ε.

fall *within* the interval. In our case $N = 10000$ and the relative ε has to intercept the probability plot at the value $1/(20 \cdot 10000)$, and this happens with $\varepsilon \approx 0.76$. We conclude that in at least 95% of the cases, *all* the points of the PD plot will stay within the interval $[-0.76, 0.76]$.

This agrees with the PD Plot shown in Figure 2.10 and proves that the PD plot cannot fluctuate too much and will therefore not capture any interaction between Y and X_2. It is important to underline that this happens because of the averaging tendency that PD plots have in this situation, rather than a possibly faulty SGB algorithmic output $\hat{Y}$.

Another issue of the PD Plot lies on the fact that this technique assumes *independence* among $\boldsymbol{x}_S$ and $\boldsymbol{x}_C$ variables. This assumption is made in Equation (2.50) when the marginal distribution does not depend on the on the fixed values $\boldsymbol{x}_S$, but is retrieved from sampling instead. As a consequence, if there is strong correlation between such subsets of variables, Equation (2.50) might fail to converge to (2.46) for however big N and the partial dependence plot fail to perform as expected. We conclude that the Partial Dependence Plots fail in case of strongly dependent variables or in case of heterogeneity in the feature effect. The second issue can be overcomed by ALE (Accumulated Local Effects) Plots, while the first one by ICE (Individual Conditional Expectation) plots. Let's see these techniques in detail.

2.7.2 Accumulated Local Effects (ALE)

In the previous Subsection we mentioned that PD Plot can perform poorly in case of strong dependence between $\boldsymbol{x}_C$ and $\boldsymbol{x}_S$ variables. As we suggested before, the issue lies on the fact that in case of strongly dependent variables, the conditional distribution $dP(\boldsymbol{x}_C|\boldsymbol{x}_S = x_S)$ would depend on the values x_S. Therefore the approximation made in Equation (2.47):

$$dP(\boldsymbol{x}_C|\boldsymbol{x}_S = x_S) \approx dP(\boldsymbol{x}_C) \tag{2.63}$$

is bad. This step is avoided when building ALE plots. This technique is described in Molnar (2019) and we quote the Author's words about this issue:

> If features of a machine learning model are correlated, the partial dependence plot cannot be trusted. The computation of a partial dependence plot for a feature that is strongly correlated with other features involves averaging predictions of artificial data instances that are unlikely in reality, and this can greatly bias the estimated feature effect.

We can see in a practical example the point in Molnar (2019)'s remark. To do so we generate two correlated variables x_1 and x_2. We use the np.random.multivariate_normal from the numpy library to generate $N = 6000$ samples according to:

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 2 \\ 0 \end{bmatrix}, \begin{bmatrix} 16 & 3.2 \\ 3.2 & 1 \end{bmatrix} \right) \tag{2.64}$$

Where $\sigma_1^2 = 16$, $\sigma_2^2 = 1$, and the correlation coefficient is set to $\rho = 0.8$. If we compare the marginal distribution of $\mathbb{P}(x_1)$ with the conditional distribution $\mathbb{P}(x_1|x_2 > 1)$, we see a lot of differences:

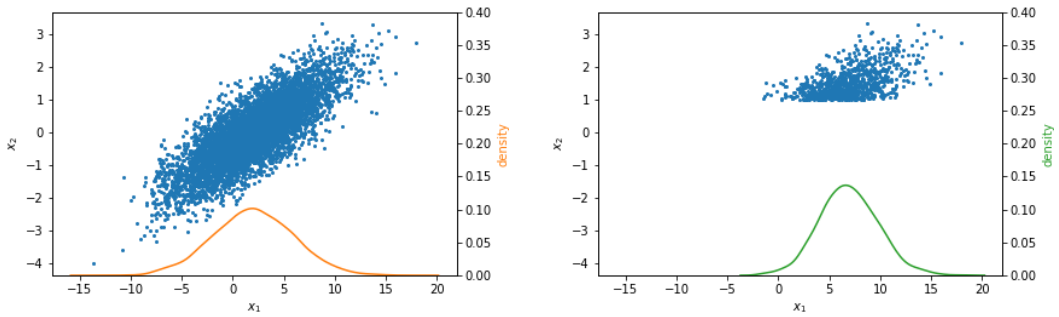

FIGURE 2.12: On the left the marginal distribution of $dP(x_1)$ is shown on the x-axis. On the right side, only the datapoints such that $x_2 > 1$ are plotted, highlighting the conditional distribution $dP(x_1|x_2 > 1)$, in green.

Note the marginal distribution of X_1 is a Gaussian $\mathcal{N}(2, 4^2)$ by construction, and the sampled data is enough to converge to the true distribution. When conditioning on $x_2 > 1$ though, the conditional distribution of x_1 has a significantly[8] higher expected mean than the marginal distribution. As we can see, the ecpted value of the conditional distribution is close to 7 and the variance has decreased.

Nevertheless, the PD plot technique will average over the original distribution and as a consequence, in our example, the higher values for x_2 will also be paired with the smallest values for x_1 (for example, the $x_1 < 0$) relatively often even if few of such combinations exists in the real data. The discrepancy in our example is quite large as one can see in Figure 2.13. In case the

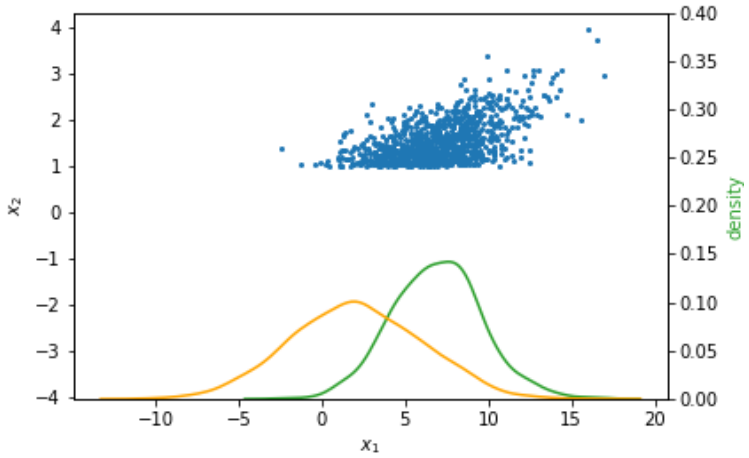

FIGURE 2.13: In green, the sampling distribution which should be used when $x_2 > 1$ in the PD Plot, in orange the area actually sampled.

reader does not want to rely solely on the (extremely) low value of the p-value and take the risk to commit a type I error with a strictly positive probability, we show a mathematical proof that the two distributions are indeed different.

Proof

We can calculate the expected value of X_1 given $X_2 > 1$ in our previous example.

We note that for the bi-variate case the $X_1|X_2 = x_2$ is known and is given by the formula:

$$X_1 \mid X_2 = x_2 \;\sim\; \mathcal{N}\left(\mu_1 + \frac{\sigma_1}{\sigma_2}\rho(x_2 - \mu_2),\ \sqrt{(1-\rho^2)}\sigma_1\right). \tag{2.65}$$

[8] The 2-samples-Kolmogorov-Smirnov statistics is around 0.30, and the related p-value is $< 10^{-200}$.

Which in our case leads to:

$$X_1 \mid X_2 = x_2 \;\sim\; \mathcal{N}\left(2 + 3.2\, x_2,\, 1.44\right). \tag{2.66}$$

Now, the distribution of $X_1 \mid X_2 > 1$ can be seen as combination of distributions of the kind $X_1 \mid X_2 = x_2$ for $x_2 \in [1, +\infty)$, where the value for x_2, each of these being weighted proportionally to the likelihood of $X_2 = x_2$ in the truncated normal distribution. We say that

$$dP(X_2 = x_2) = \frac{\frac{1}{\sqrt{2\pi}} \exp\left(-x^2/2\right) dx}{\int_1^\infty \frac{1}{\sqrt{2\pi}} \exp\left(-x^2/2\right) dx} = \frac{\frac{1}{\sqrt{2\pi}} \exp\left(-x^2/2\right) dx}{1 - \Phi(1)}$$

And the expected value for our distribution can be calculated through:

$$\mathbb{E}[X_1 \mid X_2 > 1] = \mathbb{E}[(X_1 \mid X_2 = x_2)\, dP(X_2 = x_2)] = \tag{2.67}$$

$$= \mathbb{E}\left[\mathcal{N}(2 + 3.2\, x_2, 1.44)\right] dP(X_2 = x_2) = \mathbb{E}\left[\int_1^\infty \mathcal{N}\left(2 + 3.2\, x_2, 1.44\right) dP(X_2 = x_2)\right] = \tag{2.68}$$

$$= \int_1^\infty \mathbb{E}\left[\mathcal{N}\left(2 + 3.2\, x_2, 1.44\right) dP(X_2 = x_2)\right] = \frac{\int_1^\infty (2 + 3.2\, x_2) \exp(-x^2/2)/\sqrt{2\pi}\, dx}{\int_1^\infty \exp(-x^2/2)/\sqrt{2\pi}} = \tag{2.69}$$

$$= \frac{\sqrt{2\pi}\, \mathsf{erf}(x/\sqrt{2}) - 3.2 \exp(-x^2/2)\,|_1^\infty}{1 - \Phi(1)} = \frac{\left[2\pi \mathsf{erfc}\left(\frac{1}{\sqrt{2}}\right) + 3.2\, e^{-1/2}\right] \frac{1}{\sqrt{2\pi}}}{1 - \Phi(1)} \approx 6.8785 \tag{2.70}$$

And the value ≈ 6.8785 is consistent with Figure 2.13 and it is sensibly different from the original expected value of $\mathbb{E}[X_1] = 2$ and this concludes the proof.

For a more quantitative approach on the difference between the two distributions, we need a couple of new ingredients:

- First, we need an estimate of the standard deviation of the distribution $\mathcal{N}_2$. Running the .std() buil-it Python function, we obtain a value of 2.8 ± 0.06
- Second, we need a measure for how much do these distributions differ. We will use the *Kullback-Leibler* divergence measure. Numerical methods are available for any pair of distributions, but since the analytical solution for the KL divergence between two normal distribution is available, we wil assume that $\mathcal{N}_2 \sim \mathcal{N}(6.88, 2.8^2)$

The advantage of the approximated approach is that we can easily compare two normal distributions and we can exploit the relation in (2.65) to evaluate the KL divergence as a function of the correlation coeffiient ρ. Given two normal distributions $\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $\mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$ the divergence is given by the formula in Duchi (2017):

$$D_{\mathrm{KL}}(\mathcal{N}_1 \| \mathcal{N}_2) = \frac{1}{2}\left\{\mathrm{tr}\left(\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\Sigma}_1\right) + (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^{\mathrm{T}}\, \boldsymbol{\Sigma}_2^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) - 1 + \ln\left(\frac{\det \boldsymbol{\Sigma}_1}{\det \boldsymbol{\Sigma}_2}\right)\right\}. \tag{2.71}$$

In our case $\mathcal{N}_1$ is the X_1 covariate, and we pick the conditional expectation $X_1 \mid X_2 = x_2$ for $\mathcal{N}_2$. Plugging in the expected distribution of the latter using Equation (2.65), and keeping ρ as a free parameter we get:

$$\frac{1}{2}\left[\frac{4\rho^2+1}{\sqrt{1-\rho^2}} - 1 + \frac{1}{2}\ln(1-\rho^2)\right] \tag{2.72}$$

In our specific case we chose to set $x_2 = 1$. This case is interesting because such conditional sampling is made on observation whose values for x_2 are one standard deviation higher than average. This represents events significantly above average, but still not rare (approximately 16% of the events). For such x_2 it is interesting to see ow the Kullback-Leibler divergence varies as the correlation with x_1 increases:

correlation value	K-L measure
$\rho = 0.05$	KL = 0.005
$\rho = 0.2$	KL = 0.082
$\rho = 0.35$	KL = 0.263
$\rho = 0.5$	KL = 0.583
$\rho = 0.65$	KL = 1.133
$\rho = 0.8$	KL = 2.211
$\rho = 0.9$	KL = 3.945

TABLE 2.3: KL divergence for different correlation coefficients of normally distributed random variables.

The results are not surprising: for low correlation values between X_1 and X_2, the original and the conditioned distribution do not to differ substantially and their KL divergence is low. As their correlation increases, the conditional distribution $X_1 \mid X_2 = 1$ differs more and more from the original X_1 and the KL divergence increases. Given the results from this two-dimensional and Guassian setting, we concluide that:

- Weakly correlated variables lead to a KL-divergence below 0.1.
- Average correlations lead to a KL between 0.1 and 1
- Strongly correlated variables lead to a KL above 1.

We can therefore use KL divergence as a proxy for identifying "bad sampling" in our PD Plots procedure. The advantage of this method is that it does not need to be visually plotted, and the calculation is computationally fast thanks to the `stats.entropy` function in the `scipy` Python library. Thresholds for classifying "good" vs. "average" vs. "bad" sampling in Table 2.3 are set for an explanatory purpose. We conclude that checking the KL divergence between the conditional and the unconditional distribution highlights potential failures of a PD plot. The advantage of the KL divergence lies on the fact that it does not rely on any assumption over the distributions. In order to overcome this, Molnar (2019) suggests using the Accumuated Local Effects (ALE) technique.

The ALE plots are an interesting alternative to PD Plots, the main advantage of this technique being the fact that they inference is based on a more realistic conditional distribution of the variable. On the other side, ALE plots does not focus on extrapolating predictions in case of extreme sampled events. This aspect is not desired in ABN Amro's scope, as there is need to predict extreme events as well. We therefore do not out further effort on implementing ALE plots for the FLAG model and focus on the PD Plots instead.

We conclude that when the correlation (or equivalently the KL divergence) between two variables is high, the PD Plot mis-samples the distribution and can give misleading outputs. To tackle this issue we suggest dropping on the highly correlated variables. Identifying which variables are "dependent" and correlated and which ones are the independent, is the task of the data scientist or the on-field expert.

2.7.3 Individual Conditional Expectation

The ICE plot extends the idea of the PD plots by disaggregating the average and showing the estimated functional relationship for each observation of the dataset. The concept is very simple (indeed, it is sufficient *not* to take the average while building a PD plot) but it has many advantages.

The most relevant progress of ICE plot is visible when plotting the relationship between Y and X_2 of the usual model in Equation (2.51). From the ICE plot it is indeed clear that the fitted values for Y are related to X_2, even thought the PD Plot (plotted for comparison in yellow) is flat. The results are shown in Figure 2.14.

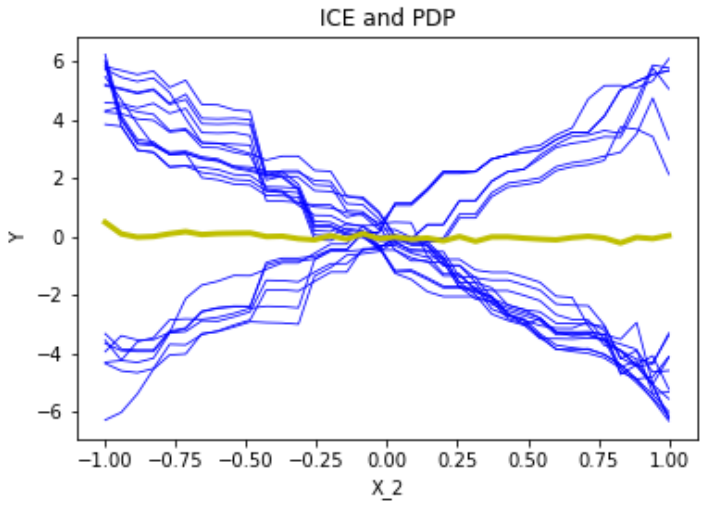

FIGURE 2.14: Response of the SGB while varying X_2 over the domain. The PD Plot, average of the ICE trajectories, is shown in yellow. Two sub-trends, one increasing and one decreasing, are visible.

The Figure shows that the ICE-built trajectories over X_2 can be split in two response groups: one group increases the $Y-$response by roughly 5 units for every unit increase of X_2 and the other decreases at the same pace. This suggests that the response from X_2 is influenced by some external factor, usually due to a variable interaction. As a comparison the PD plot is flat because of the averaging effect described in page 39.

Let us now generate the ICE plots related to the other variables, the results are shown in Figure 2.15.

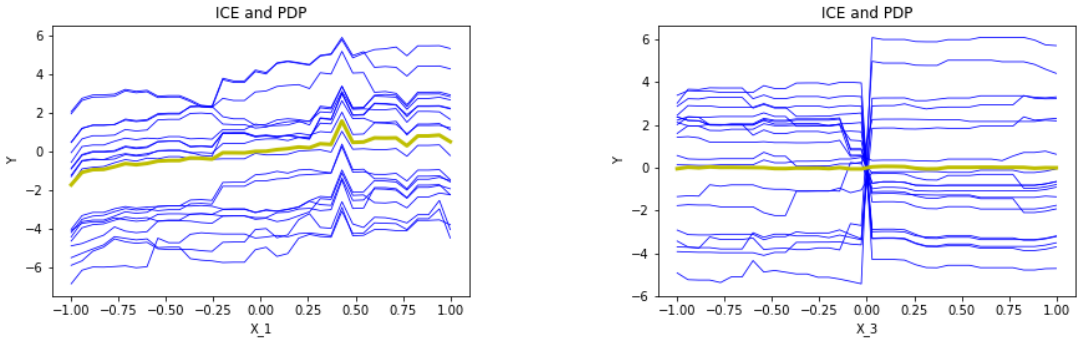

FIGURE 2.15: To the left, ICE plot along X_1, all trajectories are approximately parallel. To the right the ICE plot along X_3 shows increasing and decreasing steps around zero, suggesting an interaction.

From these plots, we can appreciate how:

- The trajectories of ICE plots over X_1 all have an increasing trend. In particular to a unit increase of X_1 corresponds a roughly unit increase of the response Y. This is consistent with the true underlying relationship. Moreover, the trajectories are roughly parallel, suggesting no interaction with other variables are occurring. In such cases the PD Plot is able to spot the same trend.
- Also in this case the PD Plot is flat, suggesting no role for X_3 in the computation of the response Y. This is misleading, and we can see though the ICE plots a sudden jump at $X_3 = 0$. However, the trajectories overlap a lot around $X_3 = 0$ and it is not clear whether the jumps are going towards the same direction or not. We will overcome this readability issue in the next Section.

The latter comparison suggests that when the ICE trajectories are roughly parallel, no interactions are occurring with the plotted variable. On the other side, Figure 2.14 and the rightmost plot of Figure 2.15 show that when interaction incur, the ICE trajectories tend to differ from each other. In order to investigate whther this intuition is correct, Goldstein et al. (2015) suggests plotting the so called *centred Indiviual Conditional Expectation* (c-ICE).

Centred-ICE

In general, when trajectories of the ICE plots intersect and have different shapes, an interaction is being spotted by the algorithm. Goldstein et al. (2015) notices that the heterogeneity of the curves is sometimes difficult to discern, especially when the trajectories are very much stacked into each other and have a wide range of intercepts (see the rightmost plot in Figure 2.15). For this reason,

it is often a good idea to centre all these trajectories starting from a common intercept (usually zero) from some value x^* for their x_S, then plot the ICE and see how much do trajectories differ from each other.

Mathematically speaking, with C represents the set of variables of which we keep the original value fixed, and S representing the set of spanning variables, given a curve $\hat{f}^{(i)}$ in the ICE plot, we define the corresponding c-ICE curve as the following:

$$\hat{f}^{(i)}_{\text{cent}}(x^*_S) = \hat{f}^{(i)} - \mathbf{1}^T \cdot \hat{f}(x^*, x_{C_i}) \tag{2.73}$$

Where the $\mathbf{1}$ vector has the dimension q of the output $\mathcal{Y}$ (normally equal to 1) and x^* is usually the minimum observed value for the variable $\boldsymbol{x}_S$. For simplicity we assume $|S| = 1$. With this of of x^* all c-ICE curves start from the same intercept and the response at the maximum $\boldsymbol{x}_S$ value $\bar{\boldsymbol{x}}_S$ for each of the curves reflects the cumulative effect of $\boldsymbol{x}_S$ on $\hat{f}$ relative to the base case. The result is a plot that better isolates the combined effect of $\boldsymbol{x}_S$ on $\hat{f}_S$. With such centred curves, it is easy to assess whether the shape of the curve is similar or interaction between variables are present and relevant. In the first case indeed, the response curves will overlap and have more or less the same cumulative effect when setting $\boldsymbol{x}_S$ to its maximum value. On the other hand when $\boldsymbol{x}_S$ interacts with other variables, a wider range of cumulative responses will appear, and the differences in shape will be spotted easily. The most relevant c-ICE plot is shown in Figure 2.16, where the relationship between Y and X_3 (that is $S = \{3\}$) is highlighted.

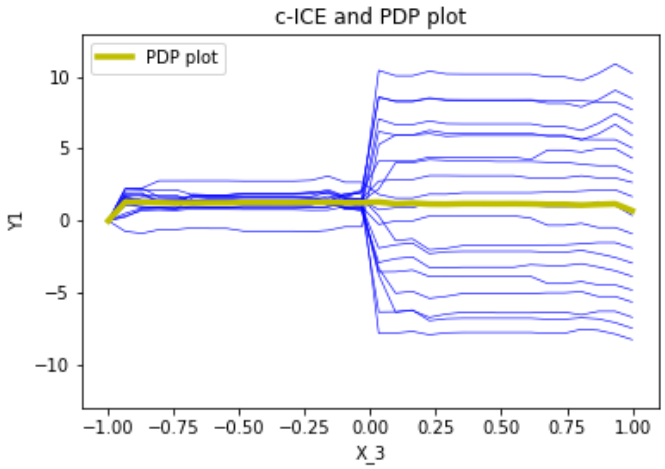

FIGURE 2.16: Caption

Unlike the ICE plots, it is clear from Figure 2.16 that the X_3 variable is interacting with another one. After centring the ICE trajectories we realise that roughly half of the jumps are positive and

the other half is negative (consistently with the flat PD plot). Comparing this Figure with the plot in Figure 2.15, we conclude that yhose instances where X_3 contributes negatively for $X_3 < 0$ start to contribute positively for $X_3 > 0$. Symmetrically, positive contributions for negative values of X_3 become (equally) negative contributions when X_3 changes sign. This suggests that X_3 is interacting with another variable (and we know it can only be X_2), and the interaction happens in a neighbourhood of X_3 changing sign. This is confirmed since the underlying model contains the $10\, X_2 \mathbf{1}_{\{X_3 \geq 0\}}$ term. Hadn't we centred the initial values, the interpretation of Figure 2.16 would ahve been more difficult. The other variabels are shown here in Figure 2.17.

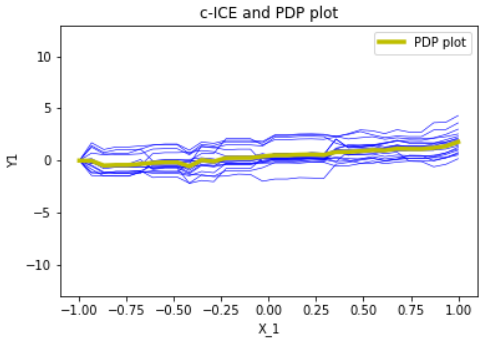

FIGURE 2.17: Centred ICE plot for X_1

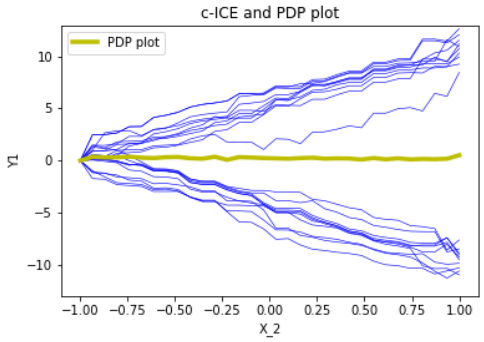

FIGURE 2.18: Centred ICE plot for X_2

As we expected, for X_1 the centred ICE plot reveals overlapping trajectories, and no interaction arises from the plot. For X_2 two sub-trends can be spotted, the cumulative effect of one of them is positive and adds up to 10 units to the prediction; the cumulative effect of the other group is negative and for $X_3 = 1$ the prediction is on average 10 units lower than for $X_3 = -1$. It is rather easy to assess to total variation of the cumulative effect of a variable on the prediction, and the bigger the variation is, the higher the effect of the interactions. We suggest a measure relate to such variability to assess the magnitude of these interactions. There is room for imagination, we propose three straightforward measures and we discuss which one is best:

- The first statistics $\mathcal{S}_1$ is defined as $\mathcal{S}_1 = \mathbb{V}ar[\hat{f}^{(i)}_{\text{cent}}(\bar{\boldsymbol{x}}_S) - \hat{f}^{(i)}_{\text{cent}}(x^*)]$, that is the variance over the dataset of the cumulative effects.
- The second candidate is $\mathcal{S}_2$, defined as $\mathcal{S}_1 = \text{range}[\hat{f}^{(i)}_{\text{cent}}(\bar{\boldsymbol{x}}_S) - \hat{f}^{(i)}_{\text{cent}}(x^*)]$, that is the difference of the maximum and minimum cumulative effect on the dataset.
- The last candidate is $\mathcal{S}_3$, defined as the difference over the top and bottom 5^{th} percentile of the cumulative effects of $\boldsymbol{x}_S$.

Now $\mathcal{S}_2$ is sensible to outliers, and $\mathcal{S}_1$ somewhat also is. We will use $\mathcal{S}_3$ in our future work.

2.7.4 Derivative-ICE

Goldstein et al. (2015) realise that "it can be difficult to visually assess derivatives from ICE plots", and points out how it can be useful to plot an estimate of the partial derivative directly. We call this a "derivative ICE" plot, in short d-ICE. Assessing derivatives of ICE curves is a helpful tool to assess the existence of interactions. In fact, when no interactions are present in the fitted model, all curves in the d-ICE plot should be equivalent and the plot roughly shows a single line. When interactions do exist, the ICE trajectories will show different shapes and the derivative lines will be therefore heterogeneous. The reasoning behind this is justified by the following: let us consider a scenario in which x_S does not interact with the other predictors. This implies that $\hat{f}$ can be written as:

$$\hat{f}(\boldsymbol{x}) = \hat{f}(\boldsymbol{x}_S, \boldsymbol{x}_C) = g(\boldsymbol{x}_S) + h(\boldsymbol{x}_C), \tag{2.74}$$

So that:

$$\frac{\partial \hat{f}(\boldsymbol{x})}{\partial \boldsymbol{x}_S} = g'(\boldsymbol{x}_S). \tag{2.75}$$

This means that in case of no interaction the d-ICE plot does not depend on the sampled values $\boldsymbol{x}_C$, and the curves are supposed to overlap. On the other hand, when interactions happen the curves will be heterogeneous and the bigger the interaction the more trajectories will differ from each other. This suggests that the standard deviation of the values of the d-ICE curves is a good indicator for interaction strength. Goldstein et al. (2015) include the value of the standard deviation of the d-ICE along the same x-axis. We think this is a sensible idea and can be a helpful tool to enhance interpretability of the plots. We therefore follow the same idea and replicate the results in the next Figure. We show the d-ICE curves obtained by the Stochastic Gradient Boosting algorithm when run over the dataset generated through (2.51).

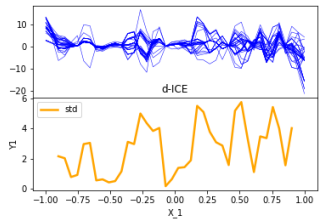

FIGURE 2.19: d-ICE plot for X_1

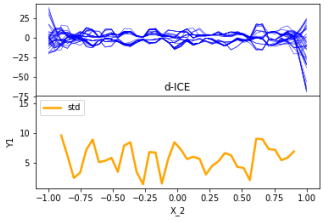

FIGURE 2.20: centred ICE plot for X_2

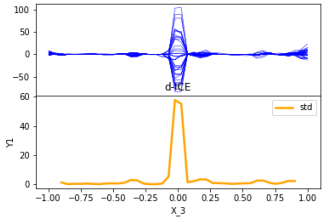

FIGURE 2.21: centred ICE plot for X_3

The plots are built by taking the numerical derivative of the ICE curves, the standard deviation of the values is taken at every discretisation point $\boldsymbol{x}_S$, and the plot is added below. The resulting plots show that X_1 has the lowest standard deviation within the d-ICE trajectories, while the d-ICE

standard deviation for X_3 peaks at $X_3 = 0$, consistent with the underlying interaction. Finally, X_2 has on average the highest deviation, consistent with the fact that it strongly interacts with X_3 all over its domain. In this toy model we also have a higher deviation for regions at the edge of the domain border because of scarcity of data and subsequent extrapolation. We therefore suggest to take into consideration the following statistics for every variable to spot interactions:

- The maximum value of the standard deviation of the d-ICE plot. Some smoothing should be applied to make this statistic discretisation-grid independent, at least partially. In our example X_3 the ideal d-ICE plot for the true underlying model would look like a two-sided-Dirac delta function with infinite standard deviation.
- The average value of the standard deviation plot. It is better to first normalise the domain and then take the numerical integral of the function. Higher averages mean the variable is interacting a lot, and the explanation of its behaviour can be more complex. Also, the edge of the domain should not be included in this calculation as the lack of data leads to extrapolation and higher variance.
- The previous point suggests that local maxima and their neighbourhoods can be either points of interaction with other variables, regions where little data is available and bad extrapolation is being made by the model. Depending on the underlying process, a peak of the standard deviation can also be a proxy for a non-linear change of the response. . In any case, it is good to spot these local maxima.

Remark. If the model response was perfect, I would expect the ICE plots of X_1 to scale with the coefficient in front of X_1. This implies that the σ is proportional to the linear coefficient, suggesting to normalise such statistic by dividing by the average value of the derivatives plot, readily derivable from the PD Plots. In practice this does not happen because of the noise, so we have to abandon this hypotesis.

2.7.5 d-ICE Flatness Detector

In this Section we propose an improvement of the d-ICE plotting technique, which we think improves readability in certain situations.

The toy model in Equation (2.51) is extremely simple, and its inputs come from the same common distribution Unif$(-1, 1)$. This is not the case of the models we will encounter in Chapter 3. Variabels including tax-rates or house age in Section 3.1, or cash flows in Section 3.2 can span over many orders of magnitude and some regions can have little data in them. These sparse regions also become more common as the number of input dimension increases because of the curse of dimensionality. As a result (we anticipate it here) the response of the SGB can be flat for large subsets of the domain. We believe that identifying these regions is another step toward better interpretability for various reasons:

- First, a tool which outputs the flat regions along a variable can easily be interpreted by an expert on the business side. No analytical background is needed to understand the concept of "flat response of the model" in a given subset, and evaluations about whether the model is performing sensibly can be made. This enhances *transparency* and evaluations about the *robustess* of the implemented method, two key requirements given by the European Commission (2019).
- Second, if the model is *not* supposed to output such a flat response, it must be because of the lack of data in the region under scrutiny. This encourages awareness on the true performance of the algorithm, enhancing transparency one more time.
- Finally, shall the extremes of the axes have a flat response, they can be ignored during the plotting procedure. Eliminating non-meaningful tails leads to higher resolution plotting in the rest of the domain. For this reason, we developed a tool who aims at identifying flat regions on the tails.

We give the following definition of "flat-tail regions" in a d-ICE plot:

Definition (Flat-Tail Region)**.** Consider the d-ICE plot over a set $S = \{i\}$, Let x_S be such variable and let $[a, b] \in \mathbb{R}$ be the range of such variable in the dataset. Let M be the top 1^{st} percentile of the absolute value of all d-ICE in the discretisation points. A *flat-tail region* is a subset $[a, \bar{a}] \cup [\bar{b}, b]$ such that at least 99% of the d-ICE values are pointwise $< 0.05\, M$. That is, at least 99% of the calculated derivatives are less a 1/20 of the top-1% derivative.

The definition might seem arbitrary. It is, but given threshold have shown to behave correctly in the scenarios in Chapter 3. The value M is selected in such a way to exclude extreme outliers but at the same time not to hide remarkable changes in the derivatives. The pointwise threshold is set to be 0.05 of M, in an attempt to mimic a human agent who is likely to ignore spikes lower than 5 mm in a plot of height 10 cm.

Chapter 3

Applications

In this chapter, we will apply the presented techniques form the previous chapter real world datasets: the Boston Housing Dataset, and the FLAG model from ABN Amro. It is important to test our findings in different scenarios, and the two we are using are complementary: the first one consists in a clean but small dataset and the SGB algorithm in use is part of an individual effort and quite straightforward; the second model consists in a real world application, built by a big corporation, and different challenges arise: the amount of data is larger but its quality is lower.

3.1 The Boston Housing Dataset

The first dataset we consider is the Boston Housing Dataset (BDH), an open source dataset available in Python's scikit-learn library. It contains information about different houses in Boston and their price. The dataset consist of 506 samples with 14 features each, and it derived from information collected by the U.S. Census Service concerning housing in the area of Boston. Massachusetts (*The Boston Housing Dataset* (2018)).

Our objective is to predict the house price using thirteen given features to predict the medv variable, that is the median value of the house. We train a SGB model to do this, and we comment the results using the explanatory techniques introduced in Section 2.7. Before moving any further, we dataset looks like, and what do its variables mean. The fact that the explanatory techniques are "agnostic" does not mean we can ignore how the original dataset looks like, even if no previous knowledge is assumed on the machine learning model in use.

The collected variables include socio-economical, geographical, environmental and financial factors which are expected to contribute to determine the house values. The variables are mostly continuous, the only exception being the boolean chas variable, and they have quite different ranges. Some of them have a exponential behaviour, some tend to distribute normally, others present a ceiling effect. The distribution of the house values is shown in Figure 3.1.

name	description	avg. value	std. dev.
CRIM	per capita crime rate by town	3.61	8.59
ZN	prop. of residential land zoned for lots over $2.5\,10^4$ sq. ft.	11.36	23.30
INDUS	proportion of non-retail business acres per town.	11.14	6.85
CHAS	Charles River dummy variable	0.07	0.25
NOX	nitric oxides concentration (pp10m)	0.25	0.55
RM	avg number of rooms per dwelling	6.28	0.70
AGE	proportion of owner-occupied units built prior to 1940	68.57	28.12
DIS	weighted distances to five Boston employment centres	3.80	2.10
RAD	index of accessibility to radial highways	9.55	8.70
TAX	full-value property-tax rate per $10,000	408.23	168.37
PTRATIO	pupil-teacher ratio by town	18.46	2.16
BLACK	$1000(\mathsf{Bk} - 0.63)^2$ where Bk is the proportion of blacks	356.67	91.20
LSTAT	% lower status of the population	12.65	7.13
MEDV	Median value of owner-occupied homes in $1000's	22.53	9.19

TABLE 3.1: Short name with description of the variables in the Boston Housing Dataset. Average values and standard deviations of the observations are given next to the description.

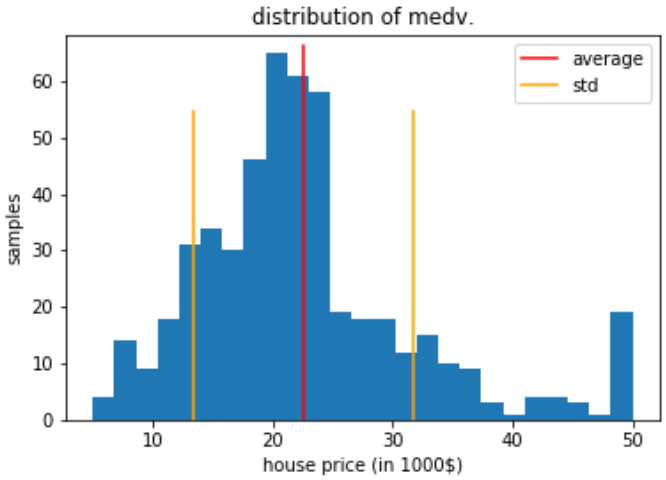

FIGURE 3.1: House value in thousand dollars. Most of the values lie between 15,000 and 30,0000 dollars. There is some ceiling effect at 50,000, we assume values have been capped for safety reasons.

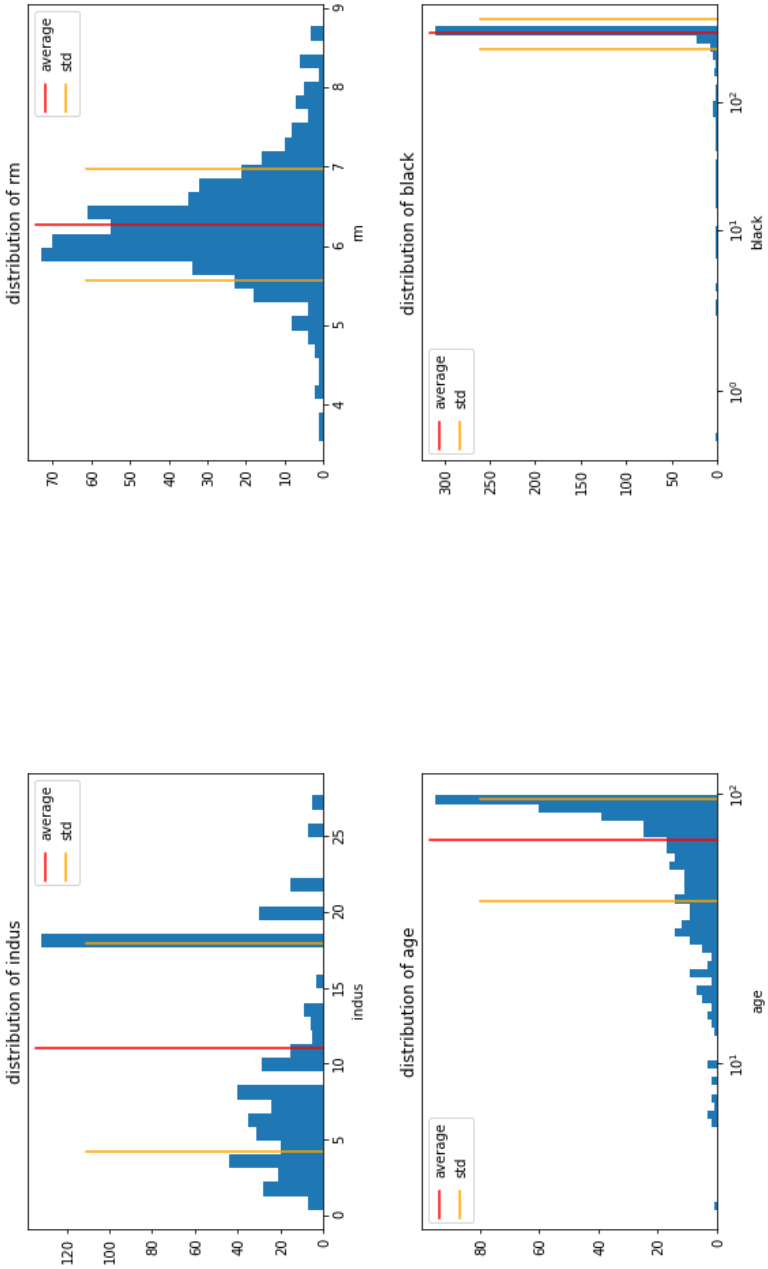

FIGURE 3.2: Histograms of the indus, rm, age and black variables. We can appreciate how rm is almost normally distributed around the value 6. age shows a ceiling effect at 100, and black has a truncated reverse exponential distribution.

Despite the relatively small size, the SGB algorithm performs well and the data quality is good. In fact, there is no need for outlier cleaning or normalising as the average performance level already reaches 0.8. In the Figure 3.3 we compare the performance before and after normalisation for different test and train sizes, and we do not spot relevant differences.

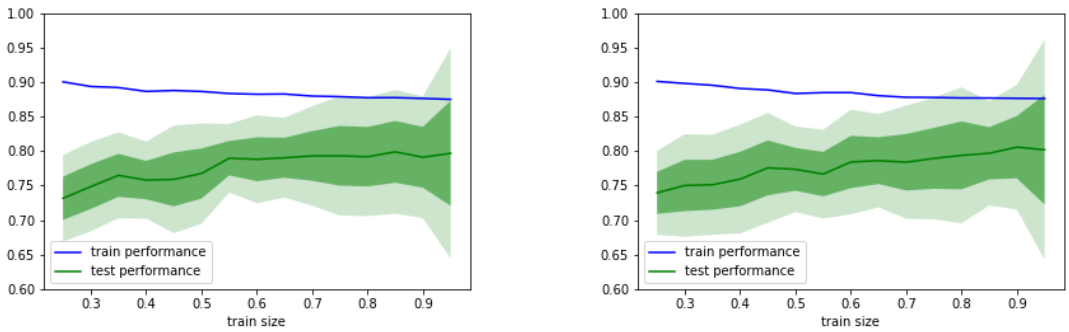

FIGURE 3.3: SGB performed prior (left) and after normalisation (right). The optimal average performance is around 0.80 in both cases and we do not see a significative difference between the two plots. Also, there is a moderate degree of overfitting which is not being mitigated by the normalisation process.

The lack of significant differences suggests that the SGB algorithm is robust under variables with different distributions, provided that such difference is not too big. In our case, a factor of approx. 1000 is the difference between the variable with the range in the smaller scale nox and the variable in the largest, black. In case of greater magnitude differences, we observed that a warning is shown by the sklearn package.

We notice that the performance levels are good, however the training and test performance do not converge perfectly around a common asymptote and show instead a discrepancy of about one σ. This shows that there is a moderate degree of overfitting. Despite tuning the parameters multiple times using a grid search the discrepancy is still there. This suggests that collecting more data would solve the problem but since we can not do this, we rather reduce the dimension of the feature space to speed up the learning process. The process of reducing the number of features is called *feature selection* and can be performed in different ways. We compare the results from the *Least Absolute Shrinkage and Selection Operator* (LASSO) and from the feature importance estimator given by a Random Forest Regressor. Both tasks are easy to implement in Python through the sklearn library and are computationally cheap given the small (13) number of features.

Remark. LASSO was introduced in order to select only a subset of covariates in regression models. Since the set of regressors is fixed in these models, using LASSO in a ML setting can fail to spot some interactions with the variables.

Remark. The drawback of RF regressor lies in its construction: this regressor splits according to variance of the response within the range of a variable, and this leads to overestimate the importance of variables with a wider range. The issue is mitigated by normalised data. Another drawback is that RF feature importance can't distinguish correlated data: it splits their importance among them, and is not able to distinguish the causal relationship between the covariates and ignore the redundant ones.

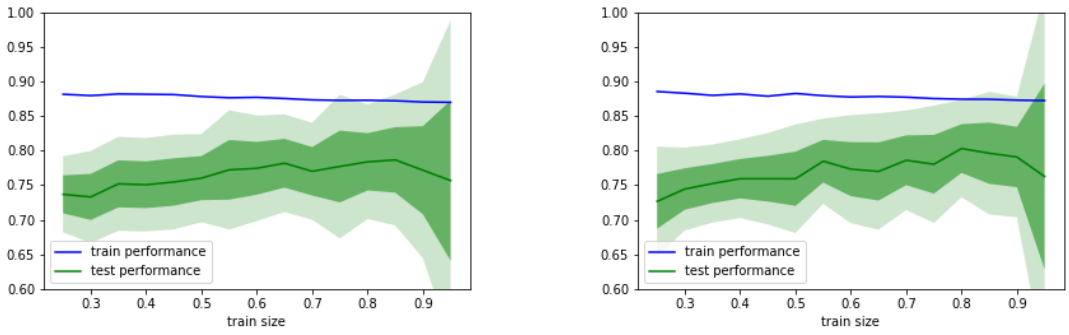

FIGURE 3.4: To the left SGB is performed after two variables have been dropped using RF's feature importance method on normalised data. To the right, SGB performance after three features are dropped using LASSO

With LASSO we eliminated indus chas nox, while with the RF method the chas, zn and rad variables had the smaller impact. However, we observe no significant increase of performance after feature selection procedures; on the other hand, we do not need to dig deeper into this issue since the problem does not affect the ABN Amro model on which we base our research.

3.1.1 Resulting PD and ICE plots

In this Subsection we show some PD plots and the ICE in action in a real world example. We introduce the so called *all-inclusive* plots where both the PD plot and the ICE plots are merged in one; in addition the projection of the datapoints on their x_S coordinates is included. This leads to better interpretability: the reader with sufficient background can see the underlying distribution, and the extrapolating power of the algorithm in regions with little data can be assessed. For readers and end-users who lack the technical knowledge, a simple PD and ICE plot should be preferred. Figures 3.5-3.8 collect for of the most interesting plots out of the Boston Housing Dataset.

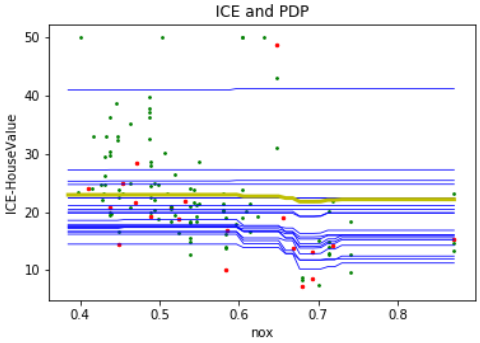

FIGURE 3.5: ICE and PD plot along the nox variable, an indicator of air pollution.

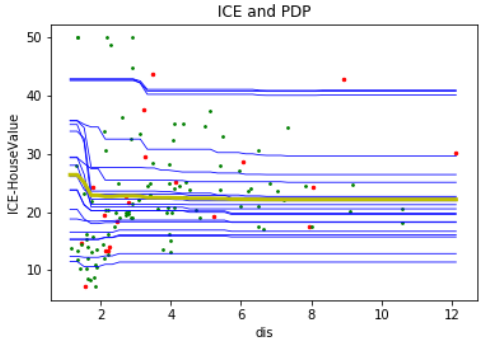

FIGURE 3.6: ICE and PD plot along the dis variable, indicating the distance from the major employment centres.

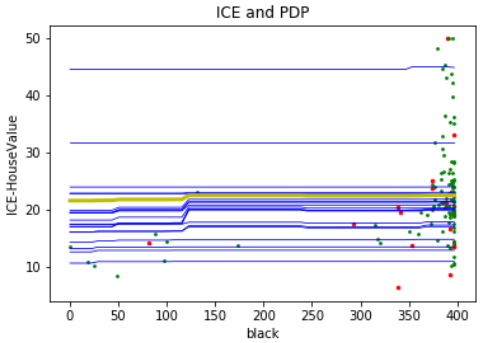

FIGURE 3.7: ICE and PD plot along the black variable, where *smaller* values stand for larger black community.

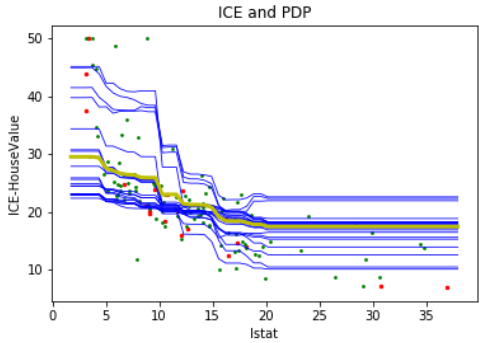

FIGURE 3.8: ICE and PD plot along the lstat variable. The higher the value, the poorer the district.

These plots show some interesting insights. Starting from the top left, we plot the expected median value of a house versus the concentration nitric oxides in the air. Nitric oxides are a relevant indicator of air pollution and we would expect a negative correlation between this variable and the output. However, there seems to little effect of this variable according to the SGB algorithm, even if a small dip can be spotted for values around 0.7 pp10m (0.7 parts per 10 million). The fact that feature selection with LASSO would drop the nox is consistent with such a flat Partial Dependency Plot. This result can be at first surprising, but an explanation to this is that such substances are

considered harmful only when the concentration values are above 10 pp10m[1], and the values in our dataset are all considered "non-harmful".

The dis variable does not show much: the response decreases slighlty as dis increases from 0 to 2 and stays flat for larger values. While this makes sense, it is surprising to see that the variable is at the same time *positively* correlated with the response. We calculated a value of $\rho = 0.31$, which shows how correlation coefficients alone are not a trustworthy measure to assess the relationship between variables.

Racial bias in Machine Learning

The black variable on the other side, is quite interesting form the ethical point of view. This is an example where the machine is learning the human racial bias and it is including it in its learning process. We can observe how the predicted house price value drops when black is lower than ≈ 125, that is when the proportion of black population is greater than 28%. Given that according to the US Census Bureau (2018), the average proportion of black population in Boston is 25.3 % , the algorithm has clearly learned to penalise neighbourhoods where the incidence is greater than average. This prediction is aligned with the existing human bias during the pricing procedure.

It is important to be aware of such biases whenever we include information related to race, gender, belief, or any possibly discriminatory attribute. The principle of non-discimination from the European Commission (2019) is clearly infringed, and any "identifiable and discriminatory bias should be removed in the collection phase where possible" (European Commission, 2019).

Finally, interesting PD and ICE plot appear when considering the lstat variable, visible in the bottom right. As we could expect, a rise of the lower status population in the neighbourhood contributes substantially to the decrease of value of the household. A quick analysis shows two interesting details:

- House value decreases as lstat increases, but when lstat> 20, no further decrease in the house value is observed (*flooring effect*). The flatness detector introduced in Section 2.7.5 can be used here.
- The decrease seems to follow more of a relative pattern than an additive one. That is, the house price decrease seems to be proportional to the house price, with most of the houses losing around 30% of their value when lstat increases from 0 to 20. We verify this finding in the next paragraph, where we introduce a new tool.

The previous bullet point raises the question whether the decreasing trend in Figure 3.8 can be modelled in absolute terms (that is, the value of the houses decreases by a certain amount of dollars), or in relative ones (that is, the house value decreases by an amount proportional to its original value). Both trends are expected to have some noise caused by the interaction of other variables, we plot the distribution and compare which one has the less dispersion. The results are shown in Figure 3.9.

[1]according to the Italian Ministry of Environment: https://www.minambiente.it/pagina/gli-inquinanti

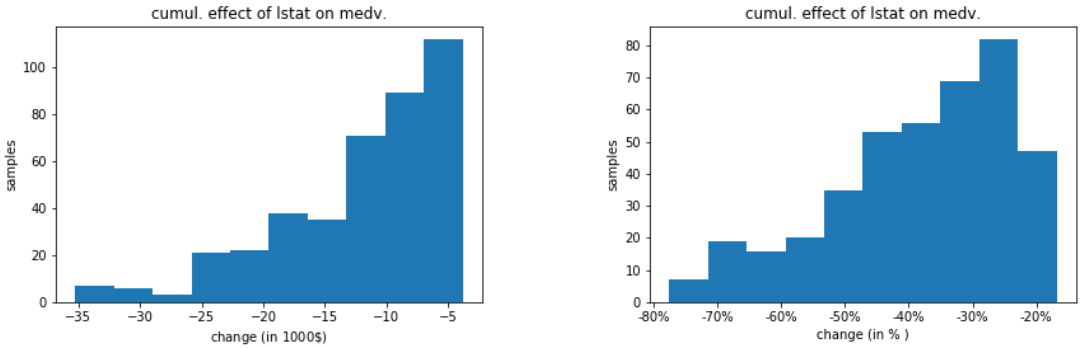

FIGURE 3.9: To the left, distribution of the cumulative effect of the lstat variabele in absolute terms. To the right, the distribution is shown in relative terms. The latter has a somewhat smaller dispersion but no further inference can be made.

In this case, we can not draw any definitive conclusion. In general, after generating the histograms for all 13 variables of the dataset, no significatn difference can be spotted. We think this is the case because:

$$\text{cumul. absolute change} = \text{cumul. relative change} \cdot \hat{f}_S(x^*). \tag{3.1}$$

And in this example all $\hat{f}_S(x^*)$ have the same order of magnitude, making it hard to distinguish the two. This will not be the case in Section 3.2, so we are confident the technique will have better discriminatory power. The door is now open for further investigation. We wonder if SGB generated models admit variables whose change sets off a multiplicative change in the output despite being a *linear* combination of weak learners. We introduce the concept of *dispersion*, and we explain how this can answer to our question.

3.1.2 Dispersion

In statistics, the dispersion of a distribution is the extent to which it is stretched or squeezed, there are many measures of of dispersion, the most common ones being the variance and standard deviation. In this work we use another measure of dispersion, the *coefficient of variation*, which is defined as:

$$c_v = \frac{\sigma}{\mu}. \tag{3.2}$$

That is, the ration between the mean of the distribution and its standard deviation. This measure has the advantage of being dimensionless and therefore scale invariant. This is an important advantage in our application since we might be obliged to normalise some data at some stage of our research.

3.1.3 d-ICE in the Boston Housing Data

Stepping back to the work of Goldstein et al. (2015), we now plot the d-ICE plots of our SGB algorithm. Goldstein et al. (2015) choose to normalise the range of the $\boldsymbol{x}_S$ variables beforehand so that the the original observations are mapped to the $[0, 1]$ interval. This guarnatees that the magnitude of the derivative of the individual expectations is linked to the relative change of the variable and does not penalise variables whose range is very wide. We remember the reader that we are computing discretised derivatives over N points, and since $\hat{f}'(x) = \frac{\Delta y}{\Delta x} = \frac{\Delta y}{\text{range}(x)/N}$, our discretised derivative decreases with the range when considering the same change of the response Δy in the regression. Plotting now the same variables we showed in Figures 3.5 - 3.8 we obtain the following plots:

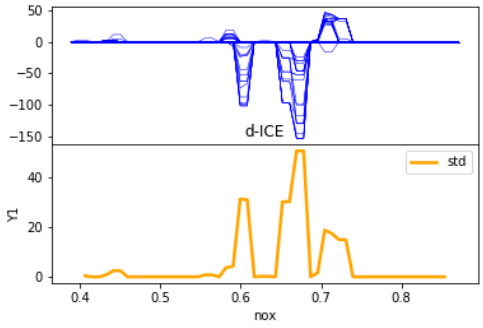

FIGURE 3.10: d-ICE plot along the nox variable, an indicator of air pollution.

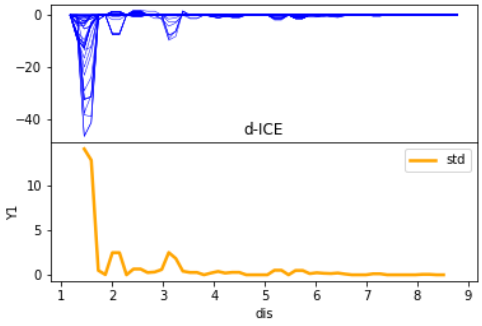

FIGURE 3.11: d-ICE plot along the dis variable.

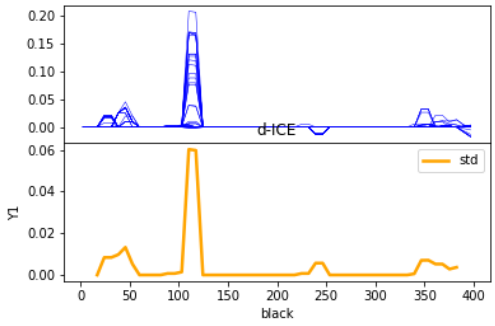

FIGURE 3.12: d-ICE plot along the black variable.

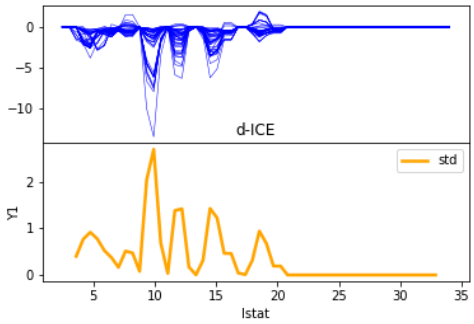

FIGURE 3.13: d-ICE plot along the lstat variable.

The points of interaction along the domain are clear. For example, we can spot two dips at nox $= 0.4$ and nox $= 0.6$ of the normalised x-axis. Using the inverse transformation we see how this corresponds to the values 0.58 and 0.68 respectively, and the result is consistent with Figure 3.5. Similarly, the black d-ICE plot shows dips at values corresponding to 125 in the original plots. The remaining two plots with lstat and dis confirm what we saw in the previous Section: dips, peaks and flat regions are where they were expected to be, provided that the user inverse-transforms the domain to compare the peaks of interaction with the All-inclusive plot. However, such comparison involves calculations which can be difficult to be made on the spot by the average user, and a comparison by eye is not possible unless both plots are available next to each other (and not even in this work they are). The whole procedure does not therefore prove to be user-friendly at all. For this reason we propose to keep the original axis when plotting the standard deviation of the derivative.

We also would like to plot only the relevant part of the plot. We make use of the flatness detector of Section 2.7.5 and we (linearly) plot only the non-flat region. In order to maintain the whole domain, we can consider plotting the remaining part in a logarithmic scale, but it is not relevant in this particular setting. Results are shown in Figures 3.14 - 3.17.

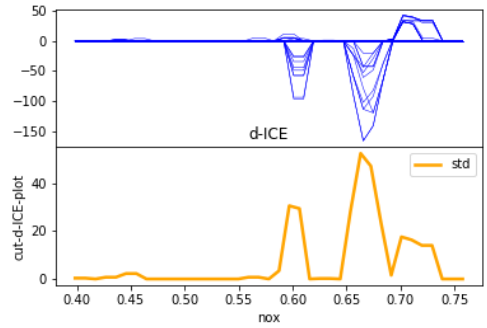

FIGURE 3.14: cut d-ICE plot along the nox variable. The algorithm cut the right tail.

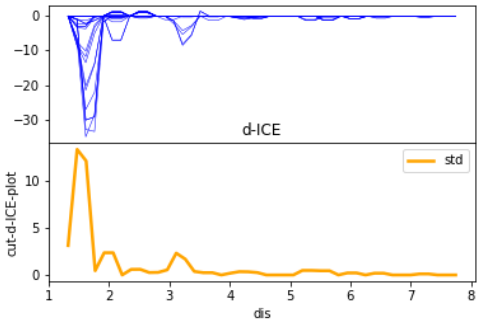

FIGURE 3.15: cut d-ICE plot along the dis variable. The rightmost values have been cut

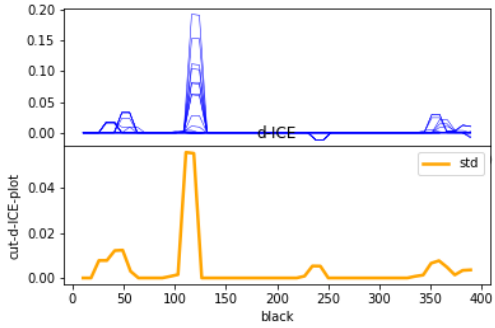

FIGURE 3.16: cut d-ICE plot along the black variable. No cut is visible

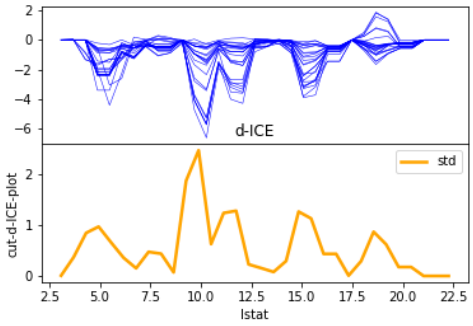

FIGURE 3.17: cut d-ICE plot along the lstat variable. All values between 22.5 and 35 have been cut.

3.1.4 d-log-ICE plots

As mentioned in Section 3.1.1, we need to consider the derivatives of the log-response to confirm or rule-out a multiplicative behaviour of a response corresponding to the peak of the standard deviation of the original derivatives.

In formulas, we are checking whether the response $\hat{f}_S$ cab be modelled as a product of non interacting sets of variables:

$$\hat{f}_S = g(\boldsymbol{x}_C)\, h(\boldsymbol{x}_S). \tag{3.3}$$

If this is the case, taking the natural logarithm on both sides leads to:

$$\log \hat{f}_S = \log g(\boldsymbol{x}_C) + \log h(\boldsymbol{x}_S) \tag{3.4}$$

And the parallelism between this and Equation 2.74 is clear. Taking the partial derivatives $\partial/\partial \boldsymbol{x}_S$ on both sides leads to:

$$\frac{\partial \log(\hat{f}_S)}{\partial \boldsymbol{x}_S} = (\log h(\boldsymbol{x}_S))' = \frac{h(\boldsymbol{x}_S)}{h'(\boldsymbol{x}_S)}, \tag{3.5}$$

which is independent from the sampled points $\boldsymbol{x}_C$, and the curves are supposed to overlap. We call this novel technique the *d-log-ICE* Plot, and we will use it on the FLAG model. We think indeed that the potential of this technique is better unlocked when the output's variable spans through different orders of magnitude: a log-scale ICE plot can better show the trends and derivatve of this plot would give the desired results.

3.2 The FLAG model

The The FR & R Loss Assessment Grade (FLAG) model has been developed in order to help the Financial Restructuring & Recovery (FR & R) department of the bank. This department, stating the internal documents of the bank Bubberman et al. (2017), is in charge of handling clients who are not longer able to meet the requirements of their credit agreement with ABN Amro, or thse who are at risk of not doing so in the near future. The FR & R department therefore undertakes efforts to either bring back clients to a performing status (whenever possible) or to recover as much of the outstanding on balance as possible. This latter event is called "write-off", as in such cases it can be decided to write-off some part of the outstanding debt on balance. The model aims at helping FR & R employees in monitoring clients by detecting early signs of risk of a "write-off" taking place, so that the client can be contacted and helped on time. In particular the model predicts the likelihood that a write-off will occur in the next six months.

3.2.1 The model

The final version of the FLAG model consists in a SGB algorithm on a dataset where 10 variables (or "risk drivers") have been selected. The variables and the relative description can be found in Table B.1 in Appendix B.1. These variables were selected among hundreds many through a Gradient Boosting procedure, and we will refer to them as *risk drivers* from now on.

The data collection includes all ABN Amro clients who are directed to the FR & R department and for whom there are at least 3 months of transaction data. When this is the case, a screenshot of the client's transaction data is taken and the values of the risk drivers are recovered. After a period of 6 months, if the client is still at FR & R, a new screenshot of the client is included. The event "write off occurred: yes / no" is registered at the same time. As a consequence, more datapoints of the same client can be available, but the group-K-fold validation technique mentioned in Section 2.4.1 assures no information leakage between the training and the validation set. The resulting data

FIGURE 3.18: Distribution of the predicted probabilities of write off. The overwhelming majority of low-probability clients forced us to plot the y-values on a log scale. Also the x-axis is plotted on a log-scale.

quality is good, and there are no missing values when running the dataframe.isnull().sum() command in Python. A summary of the main statistics of the data distribution would help for future work, but is commented out (*omissis*)

name	average	σ	min	median	max

TABLE 3.2: Summary statistics of the FLAG model risk drivers

We notice that the FIA CODE T10 bring no meaningful information as all of its values are equal to zero. This is due to that fact that this variable is an outdated indicator for a code used until 2007, and no such data appears anymore in the FLAG model. Consistently with this, any plot using an implementation of PD plot or ICE fails or does not appear at all, we will therefore ignore such risk driver from now on.

As for the response value, the SGB predictor shows a wide range of possible outputs. Most of them, like any healthy portfolio, have a low risk, but outliers with a predicted probability of write-off > 0.25 are also present. According to the FLAG model, the highest predicted probability is *omissis* and the lowest is *omissis* $1.04\,10^{-4}$ The distribution is shown in Figure 3.18

Another important remark from this preliminary analysis is that also the BEDRAGMUATIE related variables and the SALDOHUIDIG last cover a very wide range of values, while the other risk drivers such have a much smaller range, or are categorical features. This later aspect adds challenge when plotting wide-range data, but we tackle this issue by applying the *Bi-Symmetric Log transformation* by Webber (2012), who has three notable properties:

- Can squeeze large positive and negative values through a logarithmic-like mapping.
- Unlike the classical logarithmic function, it has a bounded derivative around zero.
- it is a smooth operator with a bounded and continuous gradient.

In our preliminary analysis we apply the Bi-Symmetric log-transformation to those variables whose range is greater than 200. We will follow the same rule when plotting the PD and the ICE plots of the risk drivers. The choice of 200 as a threshold is arbitrary, but with this setting we guarantee that the 5 categorical variables are plotted on a linear scale, while the continuous variables, having all a range greater than 800, are wisely plotted following the Bi-Symmetric Logarithmic one.

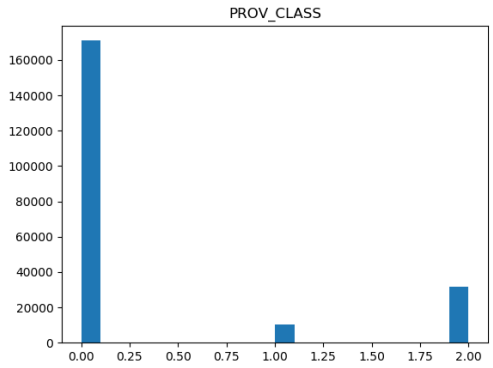

FIGURE 3.19: Example of a categorical feature, the PROV_CLASS

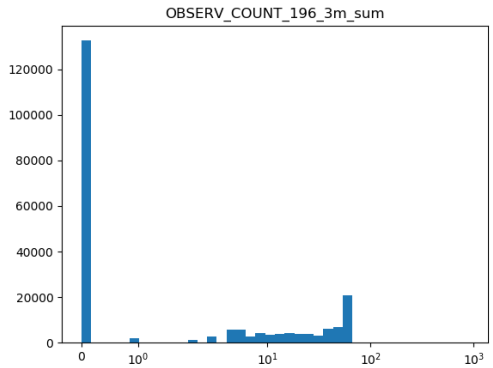

FIGURE 3.20: A variable concentrated in zero, with some data between 6 and 80.

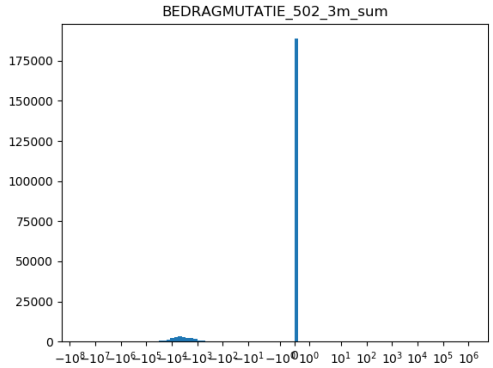

FIGURE 3.21: Risk driver with *apparently* some data quality issues.

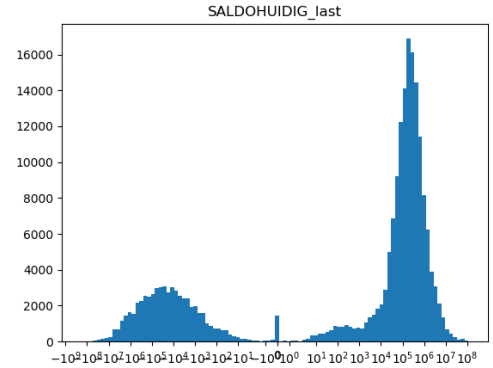

FIGURE 3.22: A clean bi-modal log-normal distribution.

The top left corner is an example of categorical feature assuming only values in $\{0, 1, 2\}$. To the top right, we see that many values of the OBERV COUNT 196 3m sum variable are equal to zero, showing no sign of of money transfer activity, as it can be the case for savings account. The remaining third of the accounts does such transactions, showing that the account is frequently used. Most of the values lie between 6 and 80, showing signs of weekly, if not daily, activity. In the bottom left, we have an example of what appears to be a "bad" risk driver: the zero values make almost 90 percent of the total and the remaining ones are well distributed between -10000 and -1000. We

wonder whether all such zero values are real observations or merely just a result of the replacement of missing data. The answer to this question is found when looking at the distribution of the transaction type among all accounts. We notice that miscellaneous payments (code 502) make less than 3% of the total transactions. Now, since two thirds of the accounts already don't register any transaction at all according to the OBSERV COUNT 196 3m sum risk driver, the BEDRAGMUTATIE 502 3m sum will be forcibly zero for them. For the remaining third, given the scarcity of transactions with such code, it is perfectly possible that no transfer with the code 502 takes place for most of the samples, and again a zero value would be assigned. This explains the large percentage of zero entries, and we will keep it as a valid risk driver.

As a contrast, the bottom-right figure shows a combination of two log-normally distributed values for the risk driver, letting appreciate us once more a glimpse of mathematical beauty in the real world. No questions arise this time about the quality of the data.

3.2.2 PD and ICE plots on FLAG

We apply the all-inclusive plot for the FLAG model. Being this a real-world model, unexpected results may appear, and the need to change or refine the current techniques may rise. We plot those risk drivers which show the plots we can learn the most from. For this reason, I will include at least one categorical variable and a continuous one, this will give the chance to test the bi-symmetric log scale in our plots.

We will omit showing the plots,

FIGURE 3.23: According to FLAG, the risk is highest when PROV CLASS= 2 and lowest when PROV CLASS = 0.

FIGURE 3.24: Plot of the risk with respect to the OBS COUNT variable.

FIGURE 3.25: Plot of the risk with respect to BEDRAGMUTATIE 3m min.

FIGURE 3.26: Probability of write-off with respect to the SALDOHUIDIG last risk driver

The plots show some interesting insights, and the behaviour of the model is mostly either increasing or decreasing with respect to each variable. The only exception in this case is the SALDOHUIDIG last risk driver, where a U-shaped behaviour is spotted.

The analysis of Figure 3.23 is relatively easy: the PROV CLASS risk driver can get values in $\{0, 1, 2\}$ so the algorithm splits the decision trees around the intermediate values 0.5 and 1.5 to distinguish the three cases. Decimal number like 0.8 or 1.3 are therefore treated as their nearest integer. This does not represent a problem as no such labels exists, while human users should be aware of the fact that filling the data-set by hand can add occasional typing mistakes and mislabel these entries. Looking now at the plot, we can see a clear trend: the risk is increasing with increasing

numeric labels of PROV CLASS, but trajectories with very low risk are compressed and it is not clear whether the jump happens for the related observations. In order to know whether this results makes sense from the business point of view we need to be familiar with the meaning of the labels.

The upper right Figure shows a decreasing trend and the scatter of the data suggests that such trend is mainly due to the higher risk associated to datapoints with BEDRAGMUTATIE 3m min equal to zero. This makes sense, as such zero observations correspond to savings accounts set up to repay mortgages, and this category is indeed more likely to incur in a loss. From this and the following two Figures we also notice how a log-scale for the y-axis would be more suitable, as most of the observations and ICE trajectories are clustered around zero.

As for Figure 3.25, the trend is quite flat, but we get better insights by plotting the logarithm of the write-off probability.

Last but not least, Figure 3.26. The trend is decreasing for negative values of the SALDOHUDIG last and becomes increasing for positive values. The trend is seen more clearly for higher starting probabilities (such as the uppermost ICE trajectory), but can be spotted on the PD plot as well. Also, it is worth noting that the pattern would not have been discovered had not we transformed the domain with a bi-symmetric log transform: the domain spans from $-2 \cdot 10^8$ to $1 \cdot 10^8$ and the locally flat region around zero spans only from $1 \cdot 10^2$ to $-1 \cdot 10^2$, making necessary more than 10^6 discretisation points to be able to spot the dip. We will check whether trajectories appear to be parallel in a log scale, and whether plotting the d-log-ICE introduced in Section 3.1.4 can give interesting results.

3.2.3 d-log-ICE plots on FLAG

Diving directly in the d-log-ICE plots can be ambitious, especially when no context is given to the user. Our approach will be to take an intermediate step and plot the ICE plots on a log scale first. We call these plots "log-ICE" plots in short. The results of the previous Section translate to Figures 3.27 - 3.30.

FIGURE 3.27: Logarithm of the risk with varying PROV CLASS.

FIGURE 3.28: Logarithm of the risk with respect to the OBS COUNT variable.

FIGURE 3.29: Logarithm of the probability of write-off with respect to the BEDRAGMUTATIE 3m min.

FIGURE 3.30: Log-Probability of write-off with respect to the SALDOHUIDIG last risk driver

As predicted, plotting with the logarithmic scale leads to simpler and better interpretation. Figure 3.27 shows how both the change in label from 0 to 1 and from 1 to 2 lead to a approximately 5-fold increase of the risk, independently from the starting level of risk. Lower-risk related values behave similarly and their trajectories are more visible here than in the linear plot.

Figure 3.28 is of a more difficult interpretation. There seems to be an interaction as higher risk observations seem have decreasing risk as OBS COUNT increases, while lower-risk observations have a risk increasing with OBS COUNT.

Similarly, we can appreciate how similarly shaped are trajectories in Figure 3.30. This leads us into finding a measure of the "depth" of the U shape, and see whether a linear or logarithmic scale gives lower dispersion. At this point, it is important to realise that these two scales are respectively linked with the absolute and the relative dispersion of the variable. In order to assess the depth of the "U"shape, we remember that if $\boldsymbol{x}_S \in [a, b]$ (the values for a and b can be found in Table 3.2), then we are interested in calculating $\hat{f}_S^i(0)$ and $\hat{f}_S^i(a)$ for every trajectory. The absolute and relative differences are taken for all i-s, their distributions are plotted, and their dispersion c_v defined in 3.1.2 is reported.

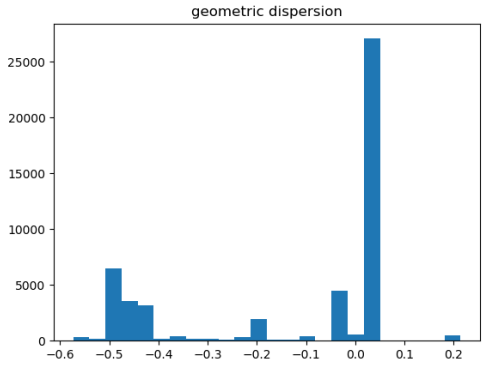

FIGURE 3.31: Distribution of accumulated relative change of SALDOHUIDIG last. There is limited variability in the data with values clustered around 0 and -0.45, and $c_v \approx -1.76$

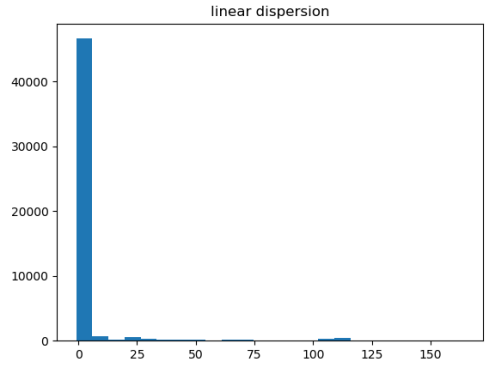

FIGURE 3.32: Distribution of the accumulated abolute change of SALDOHUIDIG last. A quick analysis shows that most of the points are clustered around -1, and a right tail with outliers is present. $c_v \approx 4.73$

We conclude that for the SALDOHUIDIG last variable when we consider the cumulated effect between the minimum value and zero, there is *less* dispersion on the relative change that on the absolute one. Moreover, the distribution of the relative change shows two distinguished clusters, one around 0 (showing "no change" in the prediction) and one around -0.45, showing that a significative amount of data has a decreased risk of about 45% when the SALDOHUIDIG last variable changes from its minimum value *omissis* to zero. Many improvements can be made on this technique, but the path is too long to follow for the scope of this work. We provide further proposals in the closing Chapter 4.

Chapter 4

Conclusions

This Thesis has been a journey through the field of Machine Learning and its purpose was to create a pleasant, readable work. The goal was to include valid mathematical applications accompanied by the awareness of the ethical challenges that the incoming raise of Artificial Intelligence is posing in our society. We pursued these two goals by focusing on the *explainable* side of Machine Learning techniques. This branch has been of great interest lately, due to the rise of powerful but hard to explain machine learning models. Among many possible approaches, we chose to focus on the paper by (Goldstein et al., 2015), where model *agnostic* visualisation Techniques are proposed. The great advantage of these techniques lies on the fact that hey do not rely on any previous knowledge of the model they try to explain, and can therefore be used in a wide range of context. This wide spectrum makes them very attractive, especially for corporations like ABN Amro which make use of many (and not necessarily ML based) models. On the other side, applying model agnostic techniques adds an extra challenge, ground-breaking results are harder to get, but we like challenges.

Concerning the ethical issues and societal stakes of the topic, the guidelines from the European Commission (2019) have been a precious source and the book from Molnar (2019) a great learning material. We highlighted the importance of explainability in credit risk, and the main points were summarised in page 4: building trust, improve the methods, identify risks.

4.1 Methodology studies

In Chapter 2 we explored the concept of PAC learnability, a fascinatingly statistical point of view on Machine Learning. The necessary background was covered in the first part of the Chapter, while the last part culminated in the introduction of the concept of weak learning and consequently a proper presentation of the AdaBoost and the Stochastic Gradient Boosting algorithms. Personally, I feel that the logical flow linking PAC learning to weak learning and eventually to (Stochastic) Gradient Boosting is worth a mention, but it doesn't blend well with the first two Sections. This happened because the theory of PAC learnability didn't prove to be a useful prevention tool to the overfitting issue. The reasons were explained at the Section 2.3.4 and can be summarised by saying that the model used as an example was not complex enough. If we could do something differently,

this would be on the list of things to change, however we are aware of the fact that simplicity is key when explaining important concepts like underfitting and overfitting.

Section 2.7 was the heart of the Chapter. The PD and the ICE plotting techniques were introduced, and the paper from (Goldstein et al., 2015) was vastly used and cited. The similar ALE technique is considered at some point, but dropped in favour of the previous two. The last pages of the Chapter were dedicated to the d-ICE from the same author and the d-ICE flatness detector. This tool is a novelty introduced in this Thesis, and can be useful in a human-machine interaction setting, its potential is great in models like FLAG, where input data can have wide ranges and the flat regions can be very large. The flatness detector limits itself to search for flat region on the extremes of the input intervals, the reason for this choice is not to jeopardise interpretability for users with non-analytical background, but it can easily modified to spot all flat regions (above certain length) in a PD or ICE plot. We suggest using the flatness detector on ICE-related plots only, as we have proved that flat PD plots are not a guarantee that the model is not "responding" to the change. For fresher details the reader can check Section 2.7.1 and the proof in page 39.

4.2 Findings in the applications

The application of the visualisation tools took place in real-world examples in Chapter 3. The first was consisted in applying a SGB method to the Boston Housing Dataset. Given the author's poor background on the housing market little inference about the model's behaviour could be made. Still, a potentially discriminatory issue was discovered in the data, it concerned the use of the black variable when predicting the house value. Although the impact of the variable was small, the example helped us understanding how even the *collection* of data for Machine Learning should be done with care. It is indeed well known that human biases can be transmitted to ML algorithms: ML algorithms learn from the data we provide them. If the data is biased, a well implemented ML algorithm will capture the same bias and its performance will benefit from it. Again we refer to the guidelines from European Commission (2019), where it is explicitly discouraged to collect potentially discriminatory data.

The second part of the Chapter focused on implementing the same techniques to the FLAG model. Here the interpretation of the model was more difficult and more important and we could not go into details as much as we wanted. However, from the talks with my colleagues at ABN it looked like the output of the model was making sense, and some new insights on how to monitor the model is given.

- First of all, input variable with both very large and very small values (abolute values) cannot be correctly visualised in a standard PD or ICE plot, and a simple log-scle on the x-axis inflates the values around zero. We propse to fix this by plotting around the 'symlog' scale in Python's matplotlib. This scale makes use of the "Bi-Symmetric Log transformation" and tackles the issue very well.

- Secondly, we believe that the d-ICE flatness detector can be a useful tool when combined with a scatterplot of the underlying datapoints (for example in Figures 3.27 - 3.30. If the

response is flat (or monotonous) and there is no data around, chances are that the model is extrapolating the output from the nearest available data and an expert should be called in order to assess if such extrapolation is correct. If on the other hand there is sufficient data in a neighbourhood of the domain there is probably not much to check.

Remark. The concept of "data in a neighbourhood of the domain" needs to be improved. The "size" of these neighbourhood is model-dependent, and analytical inference is difficult because of the curse of dimensionality. Future work can be done towards this direction, and ALE plots can be brushed up for the occasion.

4.3 Further work

In every mathematical project, regardless on the amount of work done, there is always room for more results. There have been many moments where we would have liked to stop by and elaborate further but was not possible due to time constraints (and progress pace).

One of these was after the Sections about PD plot and ICE plot (2.7.1 and 2.7.3 respectively) were completed. The idea was to implement a three-dimensional PD plotter, following the similar work by Cortez and Embrechts (2013). However, despite successful implementation for the Boston Housing Dataset model, we were not able to scale it for the FLAG model and the output plots from the first model were not interesting enough to put more effort for the FLAG model. Further work is possible in this direction.

Since the idea of three-dimensional plotting was not feasible, we also came with the idea of *reducing* a 3-d plot into a two-dimensional one. More generally, we could aim at identifying couples of important variablez for the model's response, and plot along a convex combination of them. Mathematically speaking it an be interpreted as identifying the direction $\boldsymbol{x}_v$ with the most output's variabilility (a sort of *principal component*) in the input's space $\mathbb{R}^d$. Then, consider the projection:

$$\Pi : \mathbb{R}^d \to \mathbb{R}^{q'}, \quad \text{with } q' \in \{1, 2\}. \tag{4.1}$$

That is, the projection of $\boldsymbol{x}_v$ along its top q' biggest components. We can the consider building PD and ICE plots along $\Pi(\boldsymbol{x}_v)$. It si hard to predict whther such plots can lead to more meaningful results, nor if the average user can really benefit from this.

Other research can be made on the c-ICE related statistics of 2.7.3, the question here is how strong the correlation is between the statistic and the importance of the interacting variables. The study can also link to the point-wise standard deviation of the trajectories in the d-ICE plots introduced in page 51.

Appendix A

Additional material

A.1 Known inequalities

Theorem A.1.1 (Hoeffding's Lemma)**.** *Let X be any real-valued random variable with expected value $\mathbb{E}[X] = \mu$ and $\mathbb{P}[a \leq X \leq b] = 1$. Then, for all $\lambda \in \mathbb{R}$*

$$\mathbb{E}[e^{\lambda X}] \leq \exp\left(\frac{\lambda^2(b-a)^2}{8}\right) \tag{A.1}$$

And from the previous it follows (see, cite MLT) that:

Theorem A.1.2 (Hoeffding's Inequality)**.** *Let $X_1, ..., X_m$ be a sequence of i.i.d random variables and assume that $\mathbb{E}[X_i] = \mu$ and $\mathbb{P}[a \leq X_i \leq b] = 1 \ \forall i = 1, ..., m$. Then for any $\varepsilon > 0$*

$$\mathbb{P}\left[\left|\frac{1}{m}\sum_{i=1}^{m} X_i - \mu\right| > \varepsilon\right] \leq 2\exp\left(-2m\varepsilon^2/(b-a)^2\right) \tag{A.2}$$

Theorem A.1.3. *Let X be a standard normal random variable. Then the following upper bound holds for its complementary cumulative distribution function:*

$$\Phi^c(z) = P(Z > z) < \frac{1}{\sqrt{2\pi}}\frac{1}{z}e^{-z^2/2}$$

Proof. We right down the definition of the cumulative distribution function first, and then we use the fact that $x/z > 1$ for $x > z$, we get:

$$\begin{aligned}\Phi^c(z) &= \frac{1}{\sqrt{2\pi}}\int_z^\infty e^{-x^2/2}\,dx \\ &< \frac{1}{\sqrt{2\pi}}\int_z^\infty \frac{x}{z}e^{-x^2/2}\,dx \\ &= \frac{1}{\sqrt{2\pi}}\frac{1}{z}e^{-z^2/2}.\end{aligned}$$

□

A.2 Mathematical Tools

Definition (Bi-Symmetric Log transformation)**.** Is the following mapping $\mathbb{R} \to \mathbb{R}$:

$$y = \operatorname{sgn}(x) \cdot \log_{10}(1 + |(x/C)|) \tag{A.3}$$

Note how for $|x| >> C$ we have:

$$\log_{10}(1 + |(x/C)|) \approx \log_{10}(|x/C|), \tag{A.4}$$

which means that for large x the mapping is similar to a logarithmic scale. For $|x| << C$ on the other hand we have

$$\log_{10}(1 + |(x/C)|) \approx |x/C|/\ln 10, \tag{A.5}$$

that is, roughly linear. The custom value for $C = 1/\ln 10$ gives a unity transfer function at zero and bounds the derivative at 1. Through the LogSymmTransform Python package, it is possible to introduce a linear threshold, that is, it is possible to set a symmetric region around zero for which the mapping is linear.

Theorem A.2.1 (Union Bound)**.** *For any finite or countable set of events, the probability that at least one of the events happens is no greater than the sum of the probabilities of the individual events. In formulas:*

$$\mathbb{P}\left(\bigcup_i A_i\right) \leq \sum_i \mathbb{P}(A_i) \tag{A.6}$$

Proof. The proof follows by induction. The fact that probabilities are non-negative, and associativity of the union operator are used. □

A.3 Statistical Tools

Definition. The *Coefficient of determination* R^2 of a model $f : \mathcal{X} \to \mathbb{R}$ is the proportion of the variance in the dependent variable that can be explained by the fitted model. Let $(\boldsymbol{x_1}, \ldots, \boldsymbol{x_n})$, $(y_1, \ldots, y_n)$ be the data and let $(f_1, \ldots, f_n)$ be the prediction made by the model, the coefficient of determination is defined as:

$$R^2 = 1 - \frac{\sum_i^n (y_i - f_i)^2}{\sum_i^n (y_i - \bar{y})^2} = 1 - \frac{SS_{res}}{SS_{tot}}, \tag{A.7}$$

where $\bar{y} = \frac{1}{n} \sum_i^n y_i$.

Definition (CAP curve)**.** Given a binary classification model $f : \mathcal{X} \to \{\pm 1\}$ The *Cumulative Accuracy Profile curve* curve is a measure of the discriminatory power of f. The model f first sorts the data points in increasing (predicted) order. That is, a sequence $\{-1, ..., -1, +1, ..., +1\} = \{f_1 \leq f_2 \leq ... \leq f_m\}$ is created and, without loss of generality, the input data indexes are ordered as we did. Now we define the coordinates x_i and z_i for every $i \leq m$ as the following:

$$x_i = \mathsf{rnk}(f_i)/m = i/m \tag{A.8}$$

$$z_i = \mid \{j : y_j = -1\} \mid, \tag{A.9}$$

which represents the proportion of true "-1" being selected by the first m datapoints of the model. The CAP curve $\mathcal{S}$ is defined as the piece-wise linear function passing through the set of such ordered couples $\{(0,0), (x_1, z_1), (x_2, z_2), \ldots, (x_m, z_m) = (1,1)\}$.

The curve, when compared to the curves generated by a perfect model and a random one, can then be used as a performance indicator. Such performance is called *Accuracy Rate* (AR) of the CAP curve.

Definition. The *Accuracy Rate* (AR) of a CAP curve $\mathcal{S}$ is:

$$\mathrm{CAP}_R = \frac{\int_0^1 \mathcal{S}\, dx - 0.5}{\text{Area under perfect model} - 0.5} \tag{A.10}$$

Where 0.5 is the area under the random model. Also, note that the area of a CAP curve of a perfect model for a binary classification, where the cardinality of the classes are k and $m - k$, is equal to $1 - k/2m$. Thanks to this definition the AR score is a number between 0 and 1, where 0 corresponds to the random one and 1 to the perfect model. Negative scores are theoretically possible for models which perform consistently worse than a random one, but in these cases one can simply force the model to reverse the outputs and become better than random.

Remark. The CAP curve and its Accuracy Rate can be extended to multi-class or regression models. In the multi-class case, let N be the number of classes. We can plot N CAP Curves using the "One vs All" methodology and then take the average value of the resulting N Accuracy Rates.

It is up to the modeller whether penalising some kinds of mis-classifications over others, and one can easily do so by applying a weighted average instead.

Definition. The *Gini coefficient* of a model $f : \mathcal{X} \to \mathbb{R}$ is defined as:

$$\text{Gini} = 2\,\text{CAP}_R - 1 \tag{A.11}$$

Appendix B

Related Content

B.1 Extra Tables

The table with FLAG's variable description is also omitted.

name	description

TABLE B.1: Short description of the risk drivers, based on the FLAG documentation.

B.2 Extra Figures

Here we place extra figures from ABN's model. *omissis*.

Bibliography

Ben-David, Eiron, Long, et al. (2003). "On the difficulty of approximately maximizing agreements". In: *Journal of Computer and System Sciences* 66.3, pp. 496–514.

Bubberman, Bastiaan et al. (2017). "FR& R Loss Assessment Grade Model". In: *Internal document*.

Bureau, United States Census (2018). *QuickFacts*. URL: https://www.census.gov/quickfacts/fact/table/bostoncitymassachusetts,US/PST120218.

Commission (2019). *Ethics Guidelines for Trustworthy AI*.

Cortez, Paulo and Mark J. Embrechts (Mar. 2013). "Using Sensitivity Analysis and Visualization Techniques to Open Black Box Data Mining Models". In: *Inf. Sci.* 225, pp. 1–17. ISSN: 0020-0255. DOI: 10.1016/j.ins.2012.10.039.

Duchi, John (2017). "Derivations for Linear Algebra and Optimization". In: *arXiv*, p. 13. DOI: 1706.07269..

Friedman (2001). "Greedy Function Approximation: A Gradient Boosting Machine". In: *The Annals of Statistics* 29.5, pp. 1189–1232.

Gemmell, Patrick (2018). "Research Proposal: Explaining the FLAG black box model". In: *Internal document*.

Goldstein, Alex et al. (2015). "Peeking Inside the Black Box: Visualizing Statistical Learning With Plots of Individual Conditional Expectation". In: *Journal of Computational and Graphical Statistics* 24.1, pp. 44–65. DOI: 10.1080/10618600.2014.907095.

Grover, Prince (2017). *Gradient Boosting from scratch*. URL: https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d (visited on 12/09/2017).

Hastie, Tibishirani, and Friedman (2009). *The Elements of Statistical Learning (2nd ed.)* Springer, pp. 337–384.

Jorion, Philippe (2009). "Risk Management Lessons from the Credit Risk". In: *European Financial Management*.

Marr, Bernard (2016). *What Is The Difference Between Artificial Intelligence And Machine Learning?* URL: https://www.forbes.com/sites/bernardmarr/2016/12/06/what-is-the-difference-between-artificial-intelligence-and-machine-learning/#5b780de02742.

Molnar, Christoph (2019). *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. https://christophm.github.io/interpretable-ml-book/.

Murphy, Kevin P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press.

Ribeiro et al. (2016). "Why Should I Trust You?: Explaining the Predictions of Any Classifier". In: *ArXiv*. DOI: 10.1145/2939672.2939778.

Rodriguez, Jesus (2018). *Interpretability vs. Accuracy: The Friction that Defines Deep Learning*. URL: https://towardsdatascience.com/interpretability-vs-accuracy-the-friction-that-defines-deep-learning-dae16c84db5c (visited on 05/09/2019).

Samuel, Arthur (1959). "Some Studies in Machine Learning Using the Game of Checkers". In: *IBM Journal of Research and Development* 1.3, pp. 210–229. DOI: 10.1147/rd.33.0210..

Shalev-Shwartz, Shai and Shai Ben-David (2014). *Understanding Machine Learning, from theory to algorithms*. Cambridge University Press. ISBN: 978-1-107-05713-5.

The Boston Housing Dataset (2018). URL: https://www.kaggle.com/prasadperera/the-boston-housing-dataset (visited on 06/28/2019).

Webber, J Beau W (2012). "A bi-symmetric log transformation for wide-range data". In: *Measurement Science and Technology* 24.2, p. 027001. DOI: 10.1088/0957-0233/24/2/027001.

YouTube® (2019). *YouTube hate speech policy*. URL: https://support.google.com/youtube/answer/2801939?hl=en (visited on 06/05/2019).