POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Matematica

Tesi di Laurea Magistrale

Machine Learning approach for credit score analysis

Relatrice:
Prof.ssa Patrizia Semeraro

Candidata:
Margherita Doria

Anno Accademico 2020-2021

Contents

Summary

One of the core functions of a bank is the credit risk management and one of the most important tool for it is credit score analysis. The purpose of the latter is to improve the procedure assessing creditworthiness during the credit evaluation process of a client. The foremost objective is to discriminate the lending customers on the basis of their likelihood to default, that is to identify which customers have an high likelihood of default and thus could be insolvent, and instead which customers have a lower likelihood of defaulting, being more likely to pay their financial obligations. The most commonly used credit score analysis is logit regression analysis.

In this study, we devote to use Machine Learning models in the prediction of private residential mortgage defaults. This study employs various single classification Machine Learning methodologies including Logistic Regression, K-Nearest Neighbors, Decision Trees, AdaBoost, XGBoost, Random Forest and Support Vector Machine. To further improve the predictive power, a Deep Learning technique, known as Convolutional Neural Network, widely applied to many image processing tasks, is applied to consumer credit scoring to see whether it still works well.

Two different data samples were used for the study: a public data sample and a private data sample from a Swiss bank. The study of models applied to a public data set was necessary because of the high degree of confidentiality attached to the bank's private data, and was used during the writing of the thesis to prepare and present the work carried out at the bank as an example. The private sample for this study, as we mentioned, is provided by

a private data set from a Swiss bank and an oversampling technique called SMOTE was implemented in order to treat the imbalance between classes for the response variable.
The aim of this work is to examine which method from the mentioned set exhibits the best performance in default prediction with regards to the chosen model evaluation parameters. The results on private data showed that by modelling the Deep Learning approach, we achieve a significant improvement in the predictability performance of the model. On the other hand, the results on public data showed that the model with the best predictive ability is Adaptive Boosting.
The thesis is structured as follows: the first chapter is entitled "Foundations and Basic knowledge" and presents first of all the problem of credit scoring, after which it moves on to a theoretical description of the structure of the Machine Learning models, illustrating mainly the concepts of Empirical Risk Minimization, Loss function and the Bias-Complexity trade-off. The second chapter is entitled "Theoretical presentation of the models" and its aim is a complete theoretical treatment of each model examined during the work, as well as the method of Feature Selection chosen, the treatment of the unbalancing of the data set and the metrics of evaluation of the performance of the models. The fourth and fifth chapters, entitled respectively "Application Example on public data" and "Application Example on private data" contain the description of the respective data sets and comments on the results obtained from the implementation of the models. The fifth chapter, entitled "Economic evaluation of model performance" contains considerations on the misclassification costs associated with the use of the models. Finally, the sixth chapter, "Conclusions", contains possible future work and draws conclusions from the work done.

Chapter 1

Foundations and Basic knowledge

1.1 Credit scoring economic setting

The critical role of the mortgage market in triggering several past global financial crises has led to a surge in policy interest, bank regulation and academic research in credit risk modeling. Encouraged by regulators, banks now devote significant resources in developing internal credit risk models to better quantify expected credit losses and to assign the mandatory economic capital. Rigorous credit risk analysis is not only of significance to lenders and banks but is also of paramount importance for sound economic policy making and regulation as it provides a good check on the health of a financial system and at large, the health of economies [11],[12].

For banks to be able to weight the risk of their prospective borrower being able to fulfill their repayments, they collect very specific information both on the borrower, and the underlying property of the mortgage. The outcome of these gathered data is referred to Credit Scoring, a concept emerged about 70 years ago with [13], which indicates the creditworthiness of loan applicants. These applicants are then ranked according to their credit score for the determination of their default probability and the subsequent classification into either high default likely applicant or low default likely one [14]. Banks then catalogue the gathered information to decide between providing a mortgage

or not [15]. It is important to underline the fact that behind this method of decision-making there are broader concepts such as risk aversion on the part of the financial institution, and the opportunity cost to the institution if it makes the wrong decision in providing the loan. Risk aversion is to some extent governed by the capital requirements that the bank must meet. The definition of opportunity cost, on the other hand, is the forgone benefit that would have been derived by an option not chosen. In this specific setting the option is designed as to provide or not a mortgage by the bank. So if the bank, through its credit score analysis, classifies a customer incorrectly, especially if it assigns a high probability of default to a customer who would actually have a low one, it loses the opportunity to give a loan to a performing customer.

[16] stated that "the process of modelling creditworthiness by financial institutions is referred to as credit scoring". Credit scoring is based on statistical or operational research methods. Historically, linear regression has been the most widely used techniques for building clients' scorecards. A detailed overview of credit scoring was presented by [17] including evaluation of previous published work on credit scoring and a review of discrimination and classification techniques.

The regulatory changes brought by the revised Basel Accords (subsequently adopted by national legislation in many countries and regions) introduced stronger risk management requirements for banks. The main instruments of these regulations are the minimum capital requirements, the supervisory control mechanisms and the market discipline. Under this new regulation, the capital requirements are tightly coupled to estimated credit portfolio losses. According to the Basel II/III "internal ratings-based" (IRB) approach, financial institutions are allowed to use their own internal risk measures for key drivers of credit risk as key inputs in providing loss estimates for the mortgage book and in computing capital requirements [18]. To assess the bank's credit risk exposure and provide appropriate loss estimates for the mortgage book, three risk measures are required: (i) the size of exposure at default,

(ii) the probability of default and (iii) the loss given default.
Since the 70s, regulators forced financial institutions to hold minimum capital requirements specified in the frameworks Basel I and Basel II, after which banks were motivated to adopt a forward-looking approach to determine credit risk. Nowadays, with the high availability of the enormous computational power, Machine Learning methodologies provides a framework that can be used. During the era before the highly ranked computational systems and the introduction of machine learning, credit analysts used pure judgmental approach to accept or reject applicant's form, which was tended to be based upon the view that what mattered was the 5Cs: character of the person, capital, ie how much is being asked for, collateral, ie what is the applicant willing to put up from their own resources, capacity, ie what is customer repaying ability and conditions in the market.
Traditional credit scoring models applying single-period classification techniques (e.g., logit, probit) to classify credit customers into different risk groups and to estimate the probability of default are among the most popular data mining techniques used in the industry. Classical scoring models such as the logit regression can only provide an estimate of the lifetime probability of default for a loan. However they cannot identify the existence of cures and or other competing transitions and their relationship to loan-level and macro covariates. Furthermore they do not provide insight on the timing of default, the cure from default, the time since default and time to collateral repossession ([19]).
Nowadays, because of the revolution of big data and its uncontroversial positive effect, machine learning approach, which mainly refers to a set of algorithms designed to tackle computationally intensive pattern-recognition problems in extremely large data sets, is definitely a very powerful tool. The most widely algorithms used are Bagging, Boosting, and recently Deep Learning. However, it must be kept in mind that banking institutions do not use artificial intelligence for their credit score analysis, because regulations do not allow it. This is because those types of models are black box models

that do not allow to explain why a good predictive power is obtained and therefore the most used credit score analysis tool is still logit regression.

1.2 Statistical learning framework

Machine learning is an application of Artificial Intelligence (AI) that provides systems the ability to automatically acquire a model from data or experience, basically building a learner, and then making this learner able to adapt to new data without human intervention.
In this work based on supervised learning, a binary classification task was tackled. In order to understand the meaning of these two concepts, let us briefly describe the statistical learning framework, which consists of an input, an output, a data-generation model and a measure of success.
With reference to [3] for the whole theoretical treatment, in the basic statistical learning setting, the learner has access to the following sets:

- **Domain set:** it is an arbitrary set $\mathcal{X}$ that contains the objects that we may wish to label. Usually, these domain points will be represented by a vector of features and we could refer to them as instances and to $\mathcal{X}$ as instance space.

- **Label set:** since we have dealt with supervised binary classification, we restrict the label set to be a two-element set, usually $\{0, 1\}$ or sometimes $\{-1, 1\}$. Let $\mathcal{Y}$ denote our set of possible labels. For our scopes, let $\mathcal{Y}$ be $\{0, 1\}$, where 1 represents a client who belongs to the default category and 0 a client who belongs to the non-default category.

- **Training data:** $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ is a finite sequence of pairs in $\mathcal{X} \times \mathcal{Y}$, that is a sequence of labeled domain points, often called training examples or training set. This represents the input that the learner has access to.

Then the learner is requested to output a prediction rule $h : \mathcal{X} \to \mathcal{Y}$, which is also called hypothesis or classifier and it can be used to predict the label of new domain points.
It should also be explained what is ideally the process to obtain training data. First, we assume that the instances are generated by some probability distribution over $\mathcal{X}$, whose functional form we do not know and that we call $\mathcal{D}$. It is important to highlight that the learner knows anything about this distribution. As to the labels, in our work we assume that there exists some "correct" labeling function $f : \mathcal{X} \to \mathcal{Y}$, which is unknown to the learner, and such that $y_i = f(x_i)$ for all i. This unknown labeling function is properly what the learner is trying to figure out and, summing up, we want to say that each pair in the training data S is generated by first sampling a point x_i according to $\mathcal{D}$ and then labeling it by f.
Finally, regarding the measures of success of our classifier, we define the error of a classifier to be the probability that it does not predict the correct label on a random data point generated by the aforementioned underlying distribution. Namely, the error of h is the probability to draw a random instance x, according to the distribution $\mathcal{D}$, such that $h(x)$ does not equal $f(x)$. Formally, given a domain subset $A \subset \mathcal{X}$, the probability distribution $\mathcal{D}$ assigns a number $\mathcal{D}(A)$, which determines how likely it is to observe a point $x \in A$. In many cases, we refer to A as an event and express it using a function $\pi : \mathcal{X} \to \{0, 1\}$, that is, $A = \{x \in \mathcal{X} : \pi(x) = 1\}$. In that case, we also use the notation $\mathbb{P}_{x \backsim \mathcal{D}}[\pi(x) = 1]$ to express $\mathcal{D}(A)$.
We define the error of a prediction rule, $h : \mathcal{X} \to \mathcal{Y}$, to be

$$L_{(\mathcal{D},f)}(h) \doteq \mathbb{P}_{x \backsim \mathcal{D}}[h(x) \neq f(x)] \doteq \mathcal{D}(\{x : h(x) \neq f(x)\}). \tag{1.1}$$

That is, the error of such h is the probability of randomly choosing an example x for which $h(x) \neq f(x)$. The subscript $(\mathcal{D} \backsim f)$ indicates that the error is measured with respect to the probability distribution $\mathcal{D}$ and the correct labeling function f. $L_{\mathcal{D},f}(h)$ is also called generalization loss, the true risk, or the true error of h.

1.3 Empirical Risk Minimization

As just stated, a learning algorithm receives as input a training set $\mathcal{S}$, sampled from an unknown distribution $\mathcal{D}$ and should output a predictor $h_{\mathcal{S}} : \mathcal{X} \to \mathcal{Y}$, aiming to find the $h_{\mathcal{S}}$ that minimizes the error with respect to the unknown $\mathcal{D}$ and f.

Since the learner does not know what $\mathcal{D}$ and f are, the true error is not directly available to the learner. Thus, the only error that can be calculated by the learner is the training error, that is the error the classifier incurs over the training sample:

$$L_{\mathcal{S}}(h) \doteq \frac{|\{i \in \{1, \ldots, m\} : h(x_i) \neq y_i\}|}{m}. \tag{1.2}$$

The terms training error, empirical error and empirical risk are all interchangeable and the learning paradigm of coming up with a predictor h that minimizes $L_{\mathcal{S}}(h)$ is called Empirical Risk Minimization (ERM).

Altough the ERM rule seems very natural, this approach may fail. This happens when the phenomenon of overfitting occurs. The definition of overfitting refers to the situation when our predictor has excellent performance on the training set, but its performance on the true "world" is very poor. So, intuitively, overfitting occurs when our hypothesis fits the training data "too well". Then we would find conditions under which there is guarantee that ERM does not overfit, namely conditions under which when the ERM predictor has good performance with respect to the training data, it is also higly likely to perform well over the underlying data distribution.

A common solution to rectify the ERM rule is to apply it over a restricted set of predictors, called hypothesis class and denoted by $\mathcal{H}$, where each $h \in \mathcal{H}$ is a function mapping from $\mathcal{X}$ to $\mathcal{Y}$. Such a restriction is determined before the learner sees the training data, so it should be based on some prior knowledge about the problem to be learned. For a given class $\mathcal{H}$ and a training sample $\mathcal{S}$, the $ERM_{\mathcal{H}}$ learner uses the ERM rule to choose a predictor $h \in \mathcal{H}$ with the lowest possible error over $\mathcal{S}$, that is:

$$ERM_{\mathcal{H}}(\mathcal{S}) \in \arg\min_{h \in \mathcal{H}} L_{\mathcal{S}}(h). \tag{1.3}$$

The simplest type of restriction on a class is imposing an upper bound on its size, that is imposing an upper bound on the number of predictors h in $\mathcal{H}$. In such a setting, we can make the following simplifying assumption:

Definition 1.1. (The Realizability Assumption)
There exists $h^\star \in \mathcal{H}$ such that $L_{(\mathcal{D},f)}(h^\star) = 0$. This assumption implies that with probability 1 over random samples $\mathcal{S}$, where the instances of $\mathcal{S}$ are sampled according to $\mathcal{D}$ and are labeled by f, we have $L_{\mathcal{S}}(h^\star) = 0$.

Thus the realizability assumption implies that for every ERM hypothesis we have that, with probability 1, $L_{\mathcal{S}}(h_{\mathcal{S}}) = 0$. However, we are interested in the true risk of $h_{\mathcal{S}}$, $L_{(\mathcal{D},f)}(h_{\mathcal{S}})$, rather then its empirical risk and it reasonable to assume that any guarantee on the error with respect to the underlying distribution $\mathcal{D}$ for our algorithm $h_{\mathcal{S}}$ should depend on the relationship between $\mathcal{D}$ and $\mathcal{S}$. So it is common to make the following assumption:

Definition 1.2. (The i.i.d. assumption)
The examples in the training set are independently and identically distributed (i.i.d.) according to the distribution $\mathcal{D}$. That is, if m is the size of $\mathcal{S}$, we denote this assumption with $\mathcal{S} \backsim \mathcal{D}^m$ and we mean that every x_i in $\mathcal{S}$ is freshly sampled according to $\mathcal{D}$ and then labeled according to the labeling function f.

Now we notice that since $L_{(\mathcal{D},f)}(h_{\mathcal{S}})$ depends on the training set $\mathcal{S}$, and this training set is picked by a random process, both the predictor $h_{\mathcal{S}}$ and the risk $L_{(\mathcal{D},f)}(h_{\mathcal{S}})$ are random variables. It is not realistic to expect that with full certainty $\mathcal{S}$ will suffice to direct the learner toward a good classifier from the point of view of $\mathcal{D}$, as there is always some probability δ that the sampled training data happens to be very nonrepresentative of the underlying $\mathcal{D}$. Then denoting with $1 - \delta$ the confidence parameter of our prediction and with ϵ the accuracy parameter, we address the probability to sample a training set for which $L_{(\mathcal{D},f)}(h_{\mathcal{S}})$ is not too large. The following result applies in this respect.

Corollary 1.3. Let $\mathcal{H}$ be a finite hypothesis class. Let $\delta \in (0,1)$ and $\epsilon > 0$ and let m be an integer that satisfies

$$m \geq \frac{log(|\mathcal{H}|/\delta)}{\epsilon}. \tag{1.4}$$

Then, for any labeling function f, and for any distribution $\mathcal{D}$ for which the realizability assumption holds (that is, for some $h \in \mathcal{H}$, $L_{(\mathcal{D},f)}(h) = 0$), with probability of at least $1-\delta$ over the choice of an i.i.d. sample $\mathcal{S}$ of size m, we have that for every ERM hypothesis $h_{\mathcal{S}}$ it holds that

$$L_{(\mathcal{D},f)}(h_{\mathcal{S}}) \leq \epsilon. \tag{1.5}$$

Proof. We define $\mathcal{H}_B = \{h \in \mathcal{H} : L_{(\mathcal{D},f)} > \epsilon\}$ as the subset of the "bad" hypothesis.

(1.6)

$$\begin{aligned}
\mathbb{P}[L_{(\mathcal{D},f)}(h_{\mathcal{S}}) > \epsilon] &\leq \mathbb{P}[\exists h \in \mathcal{H} : L_{\mathcal{S}}(h) = 0, L_{(\mathcal{D},f)} > \epsilon] \qquad (A \subset B \Rightarrow \mathbb{P}(A) \leq \mathbb{P}(B)) \\
&= \mathbb{P}[\exists h \in \mathcal{H}_B : L_{\mathcal{S}}(h) = 0] \qquad \text{(by definition of } \mathcal{H}_B) \\
&= \mathbb{P}\left[\bigcup_{h \in \mathcal{H}_B} \{L_{\mathcal{S}}(h) = 0\}\right] \\
&\leq \sum_{h \in \mathcal{H}_B} \mathbb{P}[L_{\mathcal{S}}(h) = 0] \qquad \text{(Union bound: } \mathbb{P}[A \cup B] \leq \mathbb{P}[A] + \mathbb{P}[B]) \\
&= \sum_{h \in \mathcal{H}_B} \prod_{i=1}^{m} \mathbb{P}[h(x_i) = y_i] \\
&\leq \sum_{h \in \mathcal{H}_B} (1-\epsilon)^m = |\mathcal{H}_B|(1-\epsilon)^m,
\end{aligned}$$

since $h \in \mathcal{H}_B \Rightarrow \mathbb{P}[h(x) \neq y] > \epsilon$. So we have obtained that

$$\begin{aligned}
\mathbb{P}[L_{(\mathcal{D},f)}(h_{\mathcal{S}}) > \epsilon] &\leq |\mathcal{H}_B|(1-\epsilon)^m \\
&\leq |\mathcal{H}|(1-\epsilon)^m \qquad (\mathcal{H}_B \subset \mathcal{H}) \\
&\leq |\mathcal{H}|e^{-\epsilon m} \qquad (1-x \leq e^{-x}) \\
&\leq |\mathcal{H}|e^{-\epsilon \frac{log(|\mathcal{H}|/\delta)}{\epsilon}} = \delta.
\end{aligned} \tag{1.7}$$

Then we can conclude that $\mathbb{P}[L_{(\mathcal{D},f)}(h_{\mathcal{S}}) \leq \epsilon] \geq 1-\delta$. □

The preceding corollary tells us that for a sufficiently large m, the $ERM_{\mathcal{H}}$ rule over a finite hypothesis class will be *probably* (with confidence $1-\delta$) *approximately* (up to an error of ϵ) correct. We can formalize this concept with the following definition of Probably Approximately Correct (PAC) learning.

Definition 1.4. (PAC Learnability)
A hypothesis class $\mathcal{H}$ is PAC learnable if there exists a function

$$m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$$

and a learning algorithm with the following property:
for every $\epsilon, \delta \in (0,1)$, for every distribution $\mathcal{D}$ over $\mathcal{X}$, and for every labeling function $f : \mathcal{X} \to \{0,1\}$, if the realizable assumption holds with respect to $\mathcal{H}, \mathcal{D}, f$, then when running the learning algorithm on $m \geq m_{\mathcal{H}}(\epsilon, \delta)$ i.i.d. samples generated by $\mathcal{D}$ and labeled by f, the algorithm returns a hypothesis h such that, with probability of at least $1-\delta$ (over the choice of the samples), $L_{(\mathcal{D},f)}(h) \leq \epsilon$.

The function $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ determines the sample complexity of learning $\mathcal{H}$, that is it tells us how many examples are required to guarantee a probably approximately correct solution. Thus, we can summarize the previous concepts in the following:

Corollary 1.5. Every finite hypothesis class is PAC learnable with sample complexity

$$m_{\mathcal{H}}(\epsilon, \delta) \leq \left\lceil \frac{\log(|\mathcal{H}|/\delta)}{\epsilon} \right\rceil .$$

We now remark the fact that our goal is to find some hypothesis $h : \mathcal{X} \to \mathcal{Y}$ that (probably approximately) minimizes the true risk $L_{\mathcal{D}}(h)$. However it exists an optimal predictor whose true risk is equal to zero : this is the Bayes classifier.

Definition 1.6. (The Bayes Optimal Predictor)
Given any probability distribution $\mathcal{D}$ over $\mathcal{X} \times \{0,1\}$, the best label predicting

function from $\mathcal{X}$ to $\{0,1\}$ will be:

$$f_{\mathcal{D}}(x) = \begin{cases} 1 & \text{if } \mathbb{P}[y=1|x] \geq 1/2 \\ 0 & \text{otherwise.} \end{cases}$$

It can be proved that for every probability distribution $\mathcal{D}$, the Bayes optimal predictor $f_{\mathcal{D}}$ is optimal, in the sense that no other classifier, $g : \mathcal{X} \to \{0,1\}$ has a lower error. This means that, for every classifier g, we have $L_{\mathcal{D}}(f_{\mathcal{D}}) \leq L_{\mathcal{D}}(g)$.
Although, we do not know $\mathcal{D}$, so we cannot utilize this optimal predictor $f_{\mathcal{D}}$; we have only access to the training sample. Thus, as natural consequence of this fact, we can extend the definition of PAC learnability to a more realistic, non realizable learning setup.

Definition 1.7. (Agnostic PAC learnability)
A hypothesis class $\mathcal{H}$ is agnostic PAC learnable if there exists a function $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ and a learning algorithm 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 an hypothesis h such that, with probability at least $1-\delta$ (over the choice of the m training samples),

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

Therefore, under the definition of agnostic PAC learning, a learner can still declare success if its error is not much larger than the best error achievable by a predictor from the class $\mathcal{H}$.

1.4 Generalized Loss Functions

Since in many practical problems the realizability assumption does not hold, we can relax it by replacing the "target labeling function" with a more flexible notion: a data-labels generating distribution.
For a probability distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$ we want to measure how likely

h is to make an error when labeled points are randomly drawn according to $\mathcal{D}$. Thus we can redefine, with respect to the definition given in (5.1), the true error (or risk) of a prediction rule (or hypothesis) h to be

$$L_{\mathcal{D}}(h) \doteq \mathbb{P}_{(x,y)\backsim\mathcal{D}}[h(x) \neq y] \doteq \mathcal{D}(\{(x,y) : h(x) \neq y\}). \tag{1.8}$$

We would like to find a predictor $h : \mathcal{X} \to \mathcal{Y}$ for which the true risk $L_{\mathcal{D}}(h)$ will be minimized.

In order to accommodate a wide range of learning tasks we can generalize our formalism of the measure of success in the following way. Given any set $\mathcal{H}$ (that plays the role of our hypothesis, or models) and some domain Z let l be any function from $\mathcal{H} \times Z$ to the set of the non negative real numbers, $l : \mathcal{H} \times Z \to \mathbb{R}^+$. We call such functions loss functions and in the case of prediction/classification problems we have that $Z = \mathcal{X} \times \mathcal{Y}$.

We now define the risk function to be the expected loss of a classifier $h \in \mathcal{H}$, with respect to a probability distribution $\mathcal{D}$ over Z, namely,

$$L_{\mathcal{D}}(h) \doteq \mathbb{E}_{z\backsim\mathcal{D}}[l(h,z)]. \tag{1.9}$$

We mean that we consider the expectation of the loss of h over objects z picked randomly according to $\mathcal{D}$. Similarly, we define the empirical risk to be the expected loss over a given sample $\mathcal{S} = (z_1, \ldots, z_m) \in Z^m$, namely,

$$L_{\mathcal{S}}(h) \doteq \frac{1}{m}\sum_{i=1}^{m} l(h, z_i).$$

Moreover, when we are dealing with a classification task we use the 0-1 loss, that has the following form:

$$l_{0-1}(h,(x,y)) \doteq \begin{cases} 0 & \text{if } h(x) = y \\ 1 & \text{if } h(x) \neq y. \end{cases}$$

We can note that, for a random variable α taking the values $\{0,1\}$, it holds that $\mathbb{E}_{\alpha\backsim\mathcal{D}}[\alpha] = \mathbb{P}_{\alpha\backsim\mathcal{D}}[\alpha = 1]$. Consequently, for this loss function, the definition of $L_{\mathcal{D}}(h)$ given in Equation (1.9) and Equation (1.8) coincide.

If instead we handle a prediction task the loss function is

$$l_{sq}(h,(x,y)) \doteq (h(x) - y)^2. \tag{1.10}$$

1.5 The Bias-Complexity Tradeoff

We know that unless one is careful, that training data can mislead the learner and result in overfitting. To overcome this problem, we restricted the search space to some hypothesis class $\mathcal{H}$, that can be also viewed as reflecting some prior knowledge that the learner has about the task. Although we can ask ourselves: is such prior knowledge really necessary for the success of the learning ? Maybe there exist some kind of universal learner, that is a learner without prior knowledge about a certain task, that we can use for challenging any task. In other words, we know that a specific learning task is defined by an unknown distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$ and the goal of the learner is to find a predictor $h : \mathcal{X} \to \mathcal{Y}$ whose risk $L_{\mathcal{D}}(h)$ is small enough: we are wondering if there exist a learning algorithm A and a training set size m, such that for every distribution $\mathcal{D}$, if A receives m i.i.d. samples from $\mathcal{D}$, there is a high chance it outputs a predictor h that has a low risk.
A formal way to address this question is through the No-Free-Lunch theorem, which states that no such universal learner exists. More precisely, the theorem states that for binary classification tasks, for every learner there exists a distribution on which it fails, that is it has a large risk.

Theorem 1.8. *(No-Free-Lunch)*
Let A be any learning algorithm for the task of binary classification with respect to the 0-1 loss over a domain $\mathcal{X}$. Let m be any number smaller than $|\mathcal{X}|/2$, representing a training set size. Then, there exists a distribution $\mathcal{D}$ over $\mathcal{X} \times \{0, 1\}$ such that:

1. *There exists a function $f : \mathcal{X} \to \{0, 1\}$ with $L_{\mathcal{D}}(f) = 0$.*
2. *With probability of at least 1/7 over the choice of $S \backsim \mathcal{D}^m$ we have that $L_{\mathcal{D}}(A(S)) \geq 1/8$.*

A further question we should address is the following: how does the No-Free-Lunch theorem result relate to the need for prior knowledge? Let us consider an ERM predictor over the hypothesis class $\mathcal{H}$ of all the functions

f such that $f : \mathcal{X} \to \{0, 1\}$. This class represents lack of prior knowledge, since every possible function from the domain to the label set is considered a good candidate. according to the No-Free-Lunch theorem, any algorithm that chooses its output from hypotheses in $\mathcal{H}$, and in particular the ERM predictor, will fail on some learning task. Therefore, this class is not PAC learnable, as formalized in the following corollary:

Corollary 1.9. Let $\mathcal{X}$ be an infinite domain set and let $\mathcal{H}$ be the set of all functions from $\mathcal{X}$ to $\{0, 1\}$. Then, $\mathcal{H}$ is not PAC learnable.

Proof. Assume, by way of contradiction, that the class is learnable, and choose some $\epsilon < 1/8$ and $\delta < 1/7$. by the definition of PAC learnability, there must be some learning algorithm A and an integer $m = m(\epsilon, \delta)$, such that for any data-generating distribution over $\mathcal{X} \times \{0, 1\}$, if for some function $f : \mathcal{X} \to \{0, 1\}$, $L_{\mathcal{D}}(f) = 0$, then with probability greater than $1 - \delta$ when A is applied to samples S of size m, generated i.i.d. by $\mathcal{D}$, $L_{\mathcal{D}}(A(S)) \leq \epsilon$. However, applying the No-Free-Lunch theorem, since $|\mathcal{X}| > 2m$, for every learning algorithm (and in particular for the algorithm A), there exists a distribution $\mathcal{D}$ such that with probability greater than $1/7 > \delta$, $L_{\mathcal{D}}(A(S)) > 1/8 > \epsilon$, which leads to the desired contradiction. □

Since we can escape the hazards foreseen by the No-Free-Lunch theorem by using our prior knowledge about a specific learning task to avoid the distributions that will cause us to fail when learning that task, we can express such prior knowledge by restricting our hypothesis class.

In order to choose a good hypothesis class we decompose the error of an $ERM_{\mathcal{H}}$ predictor into two components: the approximation error and the estimation error. Let $h_{\mathcal{S}}$ be an $ERM_{\mathcal{H}}$ hypothesis. Then we can write:

$$L_{\mathcal{D}}(h_S) = \epsilon_{app} + \epsilon_{est} \tag{1.11}$$

where: $\epsilon_{app} = \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h)$ and $\epsilon_{est} = L_{\mathcal{D}}(h_S) - \epsilon_{app}$.

The approximation error is the minimum risk achievable by a predictor in

the hypothesis class and it measures how much risk we have because we restrict ourselves to a specific class, that is it measures how much inductive bias we have. The approximation error does not depend on the sample size and is determined by the hypothesis class chosen. Moreover, enlarging the hypothesis class can decrease the approximation error. The estimation error, instead, is the difference between the error achieved by the ERM predictor and the approximation error. The estimation error results because the empirical risk, that is the training error, is only an estimate of the true risk, and so the predictor minimizing the empirical risk is only an estimate of the predictor minimizing the true risk. The quality of this estimation depends on both the training set size and the complexity of the hypothesis class.
Since our aim is to minimize the total risk, we face a trade off called bias-complexity trade off. On one hand, choosing $\mathcal{H}$ to be a very rich class decreases the approximation error but simultaneously might increase the estimation error, as a rich $\mathcal{H}$ might lead to overfitting. On the other hand, choosing $\mathcal{H}$ to be a very small set reduces the estimation error but might increase the approximation error, that is might lead to underfitting. Of course, a great choice for $\mathcal{H}$ is the class that contains the Bayes optimal classifier, but unfortunately this optimal classifier depends on the underlying distribution $\mathcal{D}$, which is unknown to us.

1.6 Cross Validation technique

In practice we usually have one pool of examples and we split them into three sets:

- Training set: we apply the learning algorithm with different parameters on the training set $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ to produce $\mathcal{H} = h1, \ldots, h_r$, where each model h_i contains a different value for the hyperparameters that we want our model to learn

- Validation set: we choose h^* from $\mathcal{H}$ based on the validation set and

Algorithm 1 k-Fold Cross Validation for Model Selection

1: Input: training set $S = (x_1, y1), ..., (x_m, y_m)$
learning algorithm A and a set of parameter values Θ
2: partition S into $S_1, S_2, \ldots, S_k$
3: **for** Each $\theta \in \Theta$ **do**
4: **for** $i = 1 \cdots k$ **do**
5: $h_{i,\theta} = A(S \setminus S_i; \theta)$
6: **end for**
7: $error(\theta) = \frac{1}{k} \sum_{i=1}^{k} L_{S_i}(h_{i,\theta})$
8: **end for**
9: Output: $\theta^* = \arg\min_\theta [error(\theta)]; h_{\theta^*} = A(S; \theta^*)$

Figure 1.1: K-fold Cross Validation algorithm for Model Selection.

according to the validation error

- Test set: we estimate the true error of h^* using the test set

It is important to highlight the fact that we cannot avoid the use of a test set, as the error on the validation set is negatively biased. Moreover, if the number of samples in the test set is too small we cannot reliably estimate the true error of our "best" classifier h^*, thus the training-validation-test split is to be done by choosing appropriate proportions.
One of the most widely used cross validation techniques is k-fold Cross Validation, the algorithm of which is shown in the figure 1.1. When $K = |\mathcal{S}|$ we have the Leave-One-Out Cross Validation.

Chapter 2

Theoretical presentation of the models

2.1 Logistic Regression model

This section is developed with reference to [20]. We know that if the target variable is categorical and takes only two values (binary classification task), we may think of encoding it with values 0 and 1 and applying linear regression, $Y = \beta^T \mathbf{x} + \epsilon$, where $Y \in \{0, 1\}$ in the training set.
For a new observation $\mathbf{x_0}$, the predicted value will not be 0 or 1, but this is not necessarily a issue, as we could interpret a non integer value of Y as a probability. The actual difficulty is that we might get predictions outside the interval $[0, 1]$. So in order to overcome this problem we may adopt a nonlinear transformation mapping the output into the interval $[0, 1]$. This is exactly what the logistic classifier do: it uses the logistic function

$$f(x) = \frac{e^x}{1 + e^x},$$

which is a nonlinear transformation, to express the conditional probability p_i as

$$\mathbb{P}(Y_i = 1 | X_i = x_i) = \frac{e^{\beta^T \mathbf{x_i}}}{1 + e^{\beta^T \mathbf{x_i}}} = \frac{1}{1 + e^{-\beta^T \mathbf{x_i}}}. \tag{2.1}$$

Solving the previous equation for the exponential we gain a better inside into the meaning of parameters β_j:

$$e^{\beta^T \mathbf{x}} = \frac{p_i}{1 - p_i}.$$

Then taking the logarithms of both sides and adding an error term, we obtain the statistical model

$$log\left(\frac{p_i}{1 - p_i}\right) = \beta^T \mathbf{x} + \epsilon.$$

The ratio $p_i/(1 - p_i)$ provides us with equivalent information in terms of odds. The logarithm of the odds is known as a logit function, that is linear in x in case of the logistic model.

Now we need to find a way to fit the regression coefficients against a set of observations $\mathbf{x_i}$ and $Y_i \in \{0, 1\}$, $i = 1, \ldots, n$.

The nonlinear transformation operated by the logistic function precludes the application of straightforward least squares. So we may fit the logistic regression model by maximum- likelihood.

The target variable Y_i, which can take values y_i in the set $\{0, 1\}$, may be regarded as the realization of a Bernoulli variable, whose density probability function is

$$\mathbb{P}(Y_i = y_i) = p_i^{y_i}(1 - p_i)^{1-y_i}.$$

From the expression 2.1 for the probability p_i we know that p_i depends on the regressors $\mathbf{x_i}$ and the vector of parameters β.

Assuming independence of errors, observations are independent as well, and the likelihood function is just the product of individual probabilities:

$$L = \prod_{i=1}^{n} p_i^{y_i}(1 - p_i)^{1-y_i} = \prod_{i=1}^{n} \left(\frac{1}{1 + e^{-\beta^{\mathbf{T}} \mathbf{x_i}}}\right)^{y_i} \left(\frac{e^{-\beta^{\mathbf{T}} \mathbf{x_i}}}{1 + e^{-\beta^{\mathbf{T}} \mathbf{x_i}}}\right)^{1-y_i}.$$

The task of maximizing L with respect to the vector of coefficients β can be simplified by taking its logarithm and by logarithm's properties we obtain:

(2.2)

$$\begin{aligned}
\mathcal{L} = \log L &= \sum_{i=1}^{n} \log\left(\frac{1}{1+e^{-\beta^{\mathbf{T}}\mathbf{x_i}}}\right) y_i + \sum_{i=1}^{n} \log\left(\frac{e^{-\beta^{\mathbf{T}}\mathbf{x_i}}}{1+e^{-\beta^{\mathbf{T}}\mathbf{x_i}}}\right)(1-y_i) \\
&= \sum_{i=1}^{n} y_i \left[\log\left(\frac{1}{1+e^{-\beta^{\mathbf{T}}\mathbf{x_i}}}\right) - \log\left(\frac{e^{-\beta^{\mathbf{T}}\mathbf{x_i}}}{1+e^{-\beta^{\mathbf{T}}\mathbf{x_i}}}\right)\right] + \log\left(\frac{e^{-\beta^{\mathbf{T}}\mathbf{x_i}}}{1+e^{-\beta^{\mathbf{T}}\mathbf{x_i}}}\right) \\
&= \sum_{i=1}^{n} y_i \log\left(\frac{1/1+e^{-\beta^{\mathbf{T}}\mathbf{x_i}}}{e^{-\beta^{\mathbf{T}}\mathbf{x_i}}/(1/1+e^{-\beta^{\mathbf{T}}\mathbf{x_i}})}\right) + \log\left(\frac{e^{-\beta^{\mathbf{T}}\mathbf{x_i}}}{1+e^{-\beta^{\mathbf{T}}\mathbf{x_i}}}\right) \\
&= \sum_{i=1}^{n} y_i [\log(e^{\beta^{\mathbf{T}}\mathbf{x_i}})] + \log\left(\frac{e^{-\beta^{\mathbf{T}}\mathbf{x_i}}}{1+e^{-\beta^{\mathbf{T}}\mathbf{x_i}}} \times \frac{e^{\beta^{\mathbf{T}}\mathbf{x_i}}}{e^{\beta^{\mathbf{T}}\mathbf{x_i}}}\right) \\
&= \sum_{i=1}^{n} y_i \beta^{\mathbf{T}}\mathbf{x_i} + \log\left(\frac{\mathbf{1}}{\mathbf{1}+\mathbf{e}^{\beta^{\mathbf{T}}\mathbf{x_i}}}\right) \\
&= \sum_{i=1}^{n} y_i \beta^{\mathbf{T}}\mathbf{x_i} - \log(\mathbf{1}+\mathbf{e}^{\beta^{\mathbf{T}}\mathbf{x_i}})
\end{aligned}$$

Thus the goal is to find the vector of coefficients β that maximizes this log-likelihood function, namely to solve this optimization problem:

$$\beta = \arg\max_{\beta} \mathcal{L}.$$

This is a nice optimization problem, since our objective function is convex, being combination of convex functions, as we can see rewriting the log-likelihood function in a more compact form:

$$\mathcal{L} = \log L = \sum_{i=1}^{n} \log(p_i) y_i + \sum_{i=1}^{n} \log(1-p_i)(1-y_i).$$

So we can apply the first-order optimality conditions :

$$\nabla_{\beta} \mathcal{L}_{\beta} = 0$$

and, according to the gradient descent method, obtain a solution when a minimum is reached.

2.2 K-Nearest Neighbors

This section is developed with reference to [3]. Nearest Neighbor (NN) algorithms are among the simplest of all machine learning algorithms. The idea behind the NN model is that the model needs to store all the training examples and then it tries to predict the label of any new instance on the basis of the label of its closest neighbors in the training set. The rationale behind such a method is based on the assumption that the features that are used to describe the domain points are relevant to their labelings in a way that makes close-by points likely to have the same label.
In short, the concept of the KNN algorithm is as follows: given a positive integer K and a test observation x_0, the KNN classifier first identifies the K points in the training data that are closest to x_0, represented by $\mathcal{N}_0$. It then estimates the conditional probability for class j as the fraction of points in $\mathcal{N}_0$ whose response values equal j:

$$\mathbb{P}(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} \mathcal{I}(y_i = j). \tag{2.3}$$

Finally, KNN applies Bayes rule and classifies the test observation x_0 to the class with the largest probability.
Let us now look at the KNN algorithm from a more generalized perspective. We assume that our instance domain, $\mathcal{X}$, is endowed with a metric function ρ. That is, $\rho : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a function that returns the distance between any two elements of $\mathcal{X}$. For example, if $\mathcal{X} = \mathbb{R}^d$ then ρ can be the Euclidean distance, $\rho(x, x_i) = \|x - x'\| = \sqrt{\sum_{i=1}^{d}(x_i - x_i')^2}$. However, depending on the type of data we have, there are other popular distance measures, like the Hamming distance, which computes the distance between binary vectors, the Manhattan distance, which computes the distance between real vectors using the sum of their absolute difference and finally the Minkowski distance, which is a generalization of the previous ones.

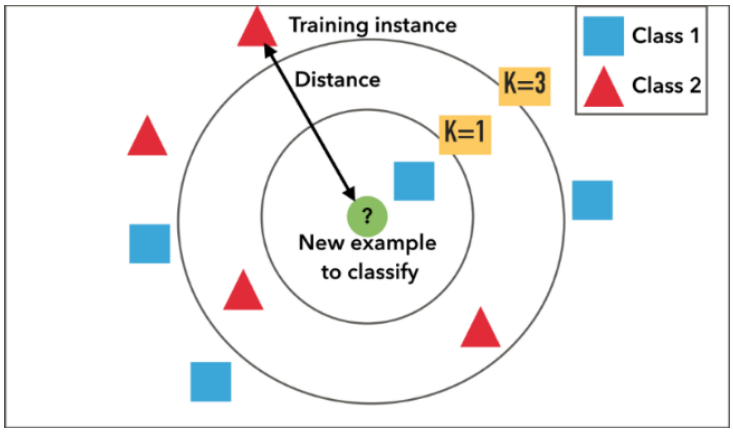

Figure 2.1: In this example we can see that the closest ($K = 1$) observation to the new one is the blue square, so the new instance will be classified as a blue square and not as a red triangle.

Let $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ be a sequence of training samples. For each $x \in \mathcal{X}$, let $\pi_1(x), \ldots, \pi_m(x)$ be a reordering of $\{1, \ldots, m\}$ (this set sometimes will be also denoted with $[m]$) according to their distance to x, $\rho(x, x_i)$. That is, $\forall i < m$,

$$\rho(x, x_{\pi_i(x)}) \leq \rho(x, x_{\pi_{i+1}(x)}).$$

For a number K, the KNN rule for binary classification is defined as follows:

input: a training sample $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$
output: for every point $x \in \mathcal{X}$,
return the majority label among $\{y_{\pi_i(x)} : i \leq k\}$.

When $k = 1$, we have the 1NN rule, represented by the classifier h:

$$h_S(x) = y_{\pi_1(x)}.$$

A geometric illustration of the 1NN rule is given in Figure 2.1.

It is also important to highlight the following aspects: the first one is that despite the fact that it is a very simple approach, KNN can often produce classifiers that are surprisingly close to the optimal Bayes classifier, even

though the true distribution is not known by the KNN classifier.
The second important fact is that the choice of K has a severe effect on the KNN classifier obtained; in fact when $K = 1$ the decision boundary is overly flexible and finds patterns in the data that do not correspond to the Bayes decision boundary. This corresponds to a classifier that has low bias but very high variance. As K grows, the method becomes less flexible and produces a decision boundary that is close to linear. This corresponds to a low-variance but high-bias classifier. Thus choosing the correct level of flexibility for the model requires to solve the bias-variance tradeoff by means, for example, cross validation.
In order to support and prove the validity of the first statement above, we now provide a finite-sample analysis of the 1NN rule, showing how the error decreases as a function of the sample size m and how it depends on properties of the distribution. We will also explain how the analysis can be generalized to KNN rules for arbitrary values of K. In particular, the analysis specifies the number of examples required to achieve a true error of $2L_D(h^*)+\epsilon$, where h^* is the Bayes optimal hypothesis.

2.2.1 A Generalization Bound for the 1NN rule

We now study the true error of the 1NN rule for binary classification task with the 0-1 loss, that is we have $Y = \{0, 1\}$ and $l(h, (x, y)) = \mathcal{I}_{[h(x)\neq y]}$. We also assume that $\mathcal{X} = [0, 1]^d$ and ρ is the Euclidean distance.
Let us introduce some notation. Let $\mathcal{D}$ be a distribution over $\mathcal{X} \times \mathcal{Y}$. Let $\mathcal{D}_\mathcal{X}$ denote the induced marginal distribution over $\mathcal{X}$ and let $\eta : \mathbb{R}^d \to \mathbb{R}$ be the conditional probability over the labels, that is

$$\eta(x) = \mathbb{P}[y = 1|x].$$

Note that from a formal point of view we have that

$$\mathbb{P}[y = 1|x] = \lim_{\delta\to 0} \frac{\mathcal{D}(\{(x', 1) : x' \in \mathcal{B}(x, \delta)\})}{\mathcal{D}(\{(x', y) : x' \in \mathcal{B}(x, \delta), y \in \mathcal{Y}\})}$$

where $\mathcal{B}(x, \delta)$ is a ball of radius δ centered around x.
We know that the Bayes optimal rule in a two-class problem, (that is, the hypothesis that minimizes $L_D(h)$ over all functions h) is

$$h^*(x) = \mathcal{I}_{[\eta(x)>1/2]}$$

which corresponds to predicting class 1 if $\mathbb{P}[y = 1|x] > 0.5$, and class 0 otherwise.
We assume that the conditional probability function η is $c-$Lipschitz for some $c > 0$: namely we mean that $\forall x, x' \in \mathcal{X}$, $|\eta(x) - \eta(x')| \leq c\|x - x'\|$. In other words, this assumption means that if two vectors (observations) are close to each other then their labels are likely to be the same.
The following Lemma applies the Lipschitzness of the conditional probability function to upper bound the true error of the 1NN rule as a function of the expected distance between each test observation and its nearest neighbor in the training set.

Lemma 2.1. *Let $\mathcal{X} = [0, 1]^d$, $\mathcal{Y} = \{0, 1\}$, and $\mathcal{D}$ a distribution over $\mathcal{X} \times \mathcal{Y}$ for which the conditional probability function η is a c-Lipschitz function.*
Let $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ be an i.i.d. sample and let h_S be its corresponding 1NN hypothesis.
Let h^* be the Bayes optimal rule for η. Then,

$$\mathbb{E}_{S\sim\mathcal{D}^m}[L_{\mathcal{D}}(h_S)] \leq 2L_{\mathcal{D}}(h^*) + c\mathbb{E}_{S\sim\mathcal{D}^m, x\sim\mathcal{D}}[\|x - x_{\pi_1(x)}\|]$$

The next step is to bound the expected distance between a random observation x and its closest element in $\mathcal{S}$. We first need the following general probability lemma, which bound the probability weight of subsets that are not hit by a random sample as a function of the size of that sample.

Lemma 2.2. *Let $C_1, \ldots, C_r$ be a collection of subsets of some domain set $\mathcal{X}$. Let S be a sequence of m points sampled i.i.d. according to some probability distribution mathcal$D$ over $\mathcal{X}$. Then,*

$$\mathbb{E}_{S\sim\mathcal{D}^m}\left[\sum_{i:\mathcal{C}_i\cap\mathcal{S}=\varnothing} \mathbb{P}[\mathcal{C}_i] \right] \leq \frac{r}{me}$$

Equipped with the preceding lemmas we are now ready to state and prove the initial statement regarding the upper bound on the expected error of the 1NN learning rule.

Theorem 2.3. *Let $\mathcal{X} = [0,1]^d$, $\mathcal{Y} = \{0,1\}$, and $\mathcal{D}$ be a distribution over $\mathcal{X} \times \mathcal{Y}$ for which the conditional probability function η ia s c-Lipschitz function. Let h_S denote the result of applying the 1NN rule to a sample $\mathcal{S} \sim \mathcal{D}^m$. Then,*

$$\mathbb{E}_{\mathcal{S}\sim\mathcal{D}^m}[L_{\mathcal{D}}(h_S)] \leq 2L_{\mathcal{D}}(h^*) + 4c\sqrt{d}m^{-\frac{1}{d+1}}.$$

Proof. Fix some $\epsilon = \frac{1}{T}$, for some $T \in \mathbb{N}$, let $r = T^d$ and let $C_1, \ldots, C_r$ be the cover of the set $\mathcal{X}$ using boxes of length ϵ: namely, for every $(\alpha_1, \ldots, \alpha_d) \in [T]^d$, there exists a set C_i of the form $\{\boldsymbol{x} : \forall j, x_j \in [(\alpha_j - 1)/T, \alpha_j/T]\}$.
We can observe that: for each $\boldsymbol{x}, \boldsymbol{x'}$ in the same box we have $\|\boldsymbol{x} - \boldsymbol{x'}\| \leq \sqrt{d}\epsilon$, otherwise, if $\boldsymbol{x}, \boldsymbol{x'}$ do not belong to the same box, we have $\|\boldsymbol{x} - \boldsymbol{x'}\| \leq \sqrt{d}$. Therefore,

$$\mathbb{E}_{\boldsymbol{x},\mathcal{S}}[\|\boldsymbol{x} - \boldsymbol{x}_{\boldsymbol{\pi_1(x)}}\|] \leq \mathbb{E}_{\mathcal{S}}\left[\mathbb{P}\left[\bigcup_{i:C_i\cap\mathcal{S}=\emptyset} C_i\right]\sqrt{d} + \mathbb{P}\left[\bigcup_{i:C_i\cap\mathcal{S}\neq\emptyset} C_i\right]\epsilon\sqrt{d}\right]$$

and by combining the result of Lemma 2.2 with the trivial bound

$$\mathbb{P}[\cup_{i:C_i\cap\mathcal{S}\neq\emptyset} C_i] \leq 1$$

and by the linearity of the expected value we get that

$$\mathbb{E}_{\boldsymbol{x},\mathcal{X}}[\|\boldsymbol{x} - \boldsymbol{x}_{\boldsymbol{\pi_1(x)}}\|] \leq \sqrt{d}\left(\frac{r}{me} + \epsilon\right).$$

Since the number of boxes is $r = T^d = (1/\epsilon)^d$ we get that

$$\mathbb{E}_{\boldsymbol{x},\mathcal{X}}[\|\boldsymbol{x} - \boldsymbol{x}_{\boldsymbol{\pi_1(x)}}\|] \leq \sqrt{d}\left(\frac{2^d\epsilon^{-d}}{me} + \epsilon\right).$$

Combining the preceding result with Lemma 2.1 we obtain that

$$\mathbb{E}_{\mathcal{S}}[L_{\mathcal{D}}(h_{\mathcal{S}})] \leq 2L_{\mathcal{D}}(h^*) + c\sqrt{d}\left(\frac{2^d\epsilon^{-d}}{me} + \epsilon\right).$$

Finally, setting $\epsilon = 2m^{-1/(d+1)}$ and noting that

$$\begin{aligned}\frac{2^d \epsilon^{-d}}{me} + \epsilon &= \frac{2^d 2^{-d} m^{d/d+1}}{me} + 2m^{-1/(d+1)} \\ &= m^{-1/(d+1)}(1/\epsilon + 2) \leq 4m^{-1/(d+1)}\end{aligned}$$

we conclude the proof. □

We can conclude that the theorem implies that if we first fix the data-generating distribution and then let m go to infinity, then the error of the 1NN rule converges to twice the Bayes error. The analysis could also be generalized to larger values of K, showing that the expected error of the KNN rule converges to $(1 + \sqrt{8/K})$ times the error of the Bayes classifier.

2.3 Decision Trees

This section is developed with reference to [3]. Within the classification setting, a decision tree is a predictor

$$h : \mathcal{X} \to \mathcal{Y} = \{0, 1\},$$

that predicts the label associated with an instance $\boldsymbol{x}$ by travelling from a root node of a tree to a leaf. At each node of the root-to-leaf path, the successor child is chosen on the basis of a splitting of the input space. Usually, the splitting is based on one of the features of $\boldsymbol{x}$ or on a predefined set of splitting rules and each leaf contains a specific label.

One of the main advantages of decision trees is that the resulting classifier is very simple to understand and graphically interpret, as we can see from the Figure 2.2.

We now present a general framework for growing a decision tree. We start with a tree with a single leaf (the root) and assign this leaf a label according to a majority vote among all labels over the training set. Then we perform a series of iterations: on each iteration, we examine the effect of splitting a single leaf. We define some "gain" measure that quantifies the improvement

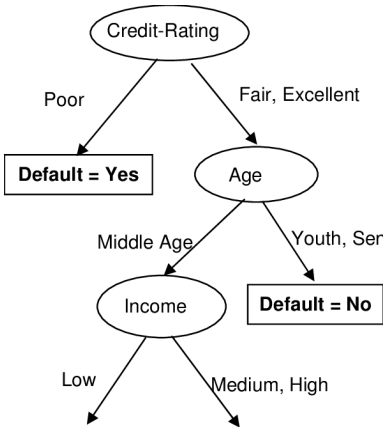

Figure 2.2: Decision Tree model to classify credit customer's creditworthiness.

due to this split. Thus, among all possible splits, we either choose the one that maximizes the gain and perform it, or choose not to split the leaf at all. In the following we provide a possible implementation, based on the popular decision tree algorithm known as "ID3" ("Iterative Dichotomizer 3").The algorithm is described for the case of binary features, namely, $\mathcal{X} = \{0,1\}^d$, and therefore all splitting rules are of the form $\mathcal{I}_{[x_i=1]}$ for some feature $i \in [d]$. Later on we also discuss the case of real valued features.
Tha algorithm works by recursive calls, with the initial call being $\texttt{ID3}(S, [d])$ and return a decision tree. In the pseudocode contained in Figure 2.3, we use a call to a procedure $\texttt{Gain}(S, i)$, which receives a training set S and an index i and evaluates the gain of a split of the tree according to the ith feature.

2.3.1 Implementations of the Gain Measure

Now we present three different implementations of $\texttt{Gain}(S, i)$.
Notation: $\mathbb{P}_S[F]$ denotes the probability that an event F holds with respect to the uniform distribution over S.
Train Error: The simplest definition of gain is the decrease in training error. Formally, let $C(a) = \min\{a, 1-a\}$. Note that the training error

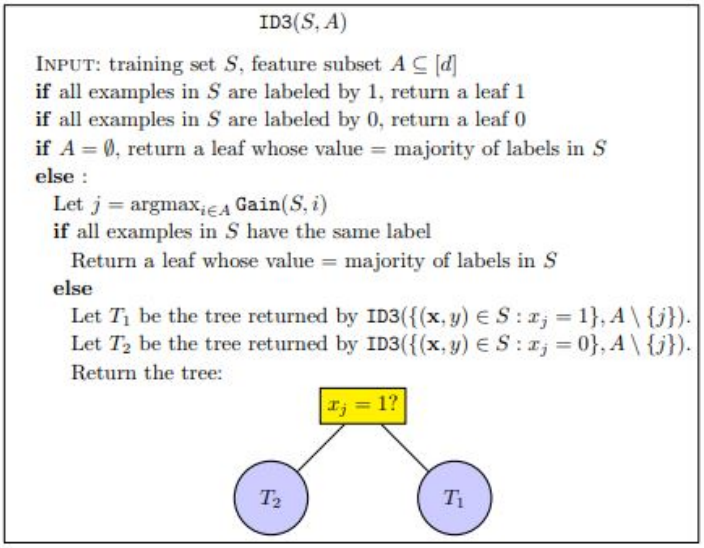

Figure 2.3: Decision Tree algorithm ID3

before splitting on feature i is

$$C(\mathbb{P}_S[y=1]) = \min\{\mathbb{P}_S[y=1], 1 - \mathbb{P}_S[y=1]\} = \min\{\mathbb{P}_S[y=1], \mathbb{P}_S[y=0]\},$$

since we took a majority vote among labels. Similarly, the error after splitting on binary feature i is

$$\mathbb{P}_S[x_i=1]C(\mathbb{P}_S[y=1|x_i=1]) + \mathbb{P}_S[x_i=0]C(\mathbb{P}_S[y=1|x_i=0]).$$

Therefore, we can define `Gain` to be the difference between the two, namley,

$$C(\mathbb{P}_S[y=1]) - \big(\mathbb{P}_S[x_i=1]C(\mathbb{P}_S[y=1|x_i=1]) + \mathbb{P}_S[x_i=0]C(\mathbb{P}_S[y=1|x_i=0])\big).$$

Information Gain: Another popular gain measure is the information gain, which is the difference between the entropy of the label before and after the split, and is achieved by replacing the function C in the previous expression by the entropy function,

$$C(a) = -a\log(a) - (1-a)\log(1-a).$$

Gini Index: Yet another definition of a gain is the Gini index, obtained by means of the following function

$$C(a) = 2a(1-a).$$

We observe that both the information gain and the Gini index are smooth and concave upper bounds of the training error. When building a classification tree, either the Gini index or the information gain are typically used to evaluate the quality of a particular split , since these two approaches are more sensitive to node purity than is the classification training error.

2.3.2 Pruning

The ID3 algorithm described previously still suffers from a big problem: the returned tree will usually be very large. Such trees may have low empirical risk, but their true risk will tend to be high.
However it holds that, with probability of at least $1-\delta$ over a sample of size m, for every n and for every decision tree $h \in \mathcal{H}$ with n nodes, we have the following bound on the true risk associated with the classifier:

$$L_D(h) \leq L_S(h) + \sqrt{\frac{(n+1)\log_2(d+3) + \log(2/\delta)}{2m}}, \tag{2.4}$$

where d is the feature space dimension.
One solution is to limit the number of iterations of ID3, leading to a tree with a bounded number of nodes. However, an other common solution is to prune the tree after is built, hoping to reduce it to a much smaller tree, but still with a similar empirical error. Theoretically, according to the bound in (2.4), if we can make n much smaller without increasing $L_{\mathcal{S}(h)}$ by much, we are likely to get a decision tree with a smaller true risk.
Usually, the pruning is performed by a bottom-up walk on the tree, in the sense that: each node might be replaced with one of its subtrees or with a leaf, based on some bound or estimate of $L_{\mathcal{D}(h)}$ (for example, we could use the bound in (2.4)).
A pseudocode of a common tree pruning procedure is given in Figure 2.4.

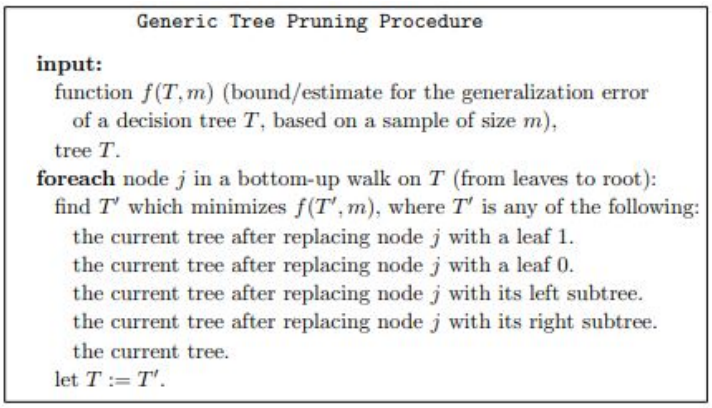

Figure 2.4: Generic tree pruning procedure.

2.3.3 Threshold-Based Splitting Rules for real-valued features

In the previous discussion we have described an algorithm for growing a decision tree assuming that the features are binary and the splitting rule are of the form $\mathcal{I}_{[x_{i=1}]}$. We can extend this result to the case of real-valued features and threshold-based splitting rules, that is, $\mathcal{I}_{[x_{i<\theta}]}$.

The main idea is to reduce the problem to the case of binary fetures as follows. Let $\boldsymbol{x_1}, \ldots, \boldsymbol{x_m}$ be the instances of the training set, whose dimension is m. for each real-valued feature i, we sort the instances so that $x_{1,i} \leq \cdots \leq x_{m,i}$. Then we define a set of thresholds $\theta_{0,i}, \ldots, \theta_{m+1,i}$ such that $\theta_{j,i} \in (x_{j,i}, x_{j+1,i})$ (with the convention $x_{0,i} = -\infty$ and $x_{m,i} = +\infty$). Finally, for each i and j we define the binary feature $\mathcal{I}_{[x_i<\theta_{j,i}]}$. Once we have constructed these binary features, we can run the ID3 algorithm escribed above. To conclude, it can be shown that for any decision tree with threshold-based splitting rules over the original real-valued features exists a decision tree over the constructed binary features with the same training error and the same number of nodes.

2.4 Random Forests

This section is developed with reference to [3]. The decision trees discussed in the previous section suffer from high variance in their results. This means that if we split the training data into two parts at random, and fit a decision tree to both halves, the results that we get could be quite different. In contrast, a procedure with low variance will yield similar results if applied repeatedly to distinct data sets; for example, linear regression tends to have low variance if the ratio of n (number of observations) and d (number of features) is moderately large.

Bootstrap aggregation, or bagging, is a general-purpose procedure for reducing the variance of a statistical learning method. In fact we know that, given a set of n independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, the variance of the mean $\bar{Z}$ of the observations is given by σ^2/n. In other words, averaging a set of observations reduces variance. Hence a natural way to reduce the variance and then increase the prediction accuracy of a statistical learning method is to take repeated samples, we say J, from our training set, build a separate prediction model using each bootstrapped training set, and take a majority vote, since we are in the classification setting. In other words, we could calculate $\hat{h}^1(x), \hat{h}^2(x), \ldots, \hat{h}^J(x)$ using J separate training sets, where $\hat{h}^j(x)$ is the classifier (in our case a classification tree) trained using the j-th bootstrapped training set, and then take a majority vote among all the J predictions. This means that for a given test observation, we can record the class predicted by each of the J trees, and take a majority vote: the overall prediction is the most commonly occurring class among the J predictions.

Random forests provide an improvement over bagged trees by way of a small tweak that decorrelates the trees. As in bagging, we build a number of decision trees on bootstrapped training samples; but when building these decision trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of d predictors. The split is allowed to use only one of those m predictors. A fresh sample of

m predictors is taken at each split, and typically we choose $m \approx \sqrt{d}$.
In other words, in building a random forests, at each split in the tree, the algorithm is not even allowed to consider a majority of the available predictors. In order to understand the reason of this concept, suppose that there is one very strong predictor in the data set, along with a number of moderately strong predictors. Then in the collection of bagged trees, most or all of the trees will use this strong predictor in the top split. Consequently, all of the bagged trees will look quite similar to each other. Hence the predictions from the bagged trees will be highly correlated. Unfortunately, taking a majority vote among many highly correlated quantities does not lead to as large of a reduction in variance as taking a majority vote among many uncorrelated quantities. In particular, this means that bagging will not lead to a substantial reduction in variance over a single tree in this setting.
Random forests overcome this problem by forcing each split to consider only a subset of the predictors. Therefore, on average $(d-m)/d$ of the splits will not even consider the strong predictor, and so other predictors will have more of a chance. We can think of this process as decorrelating the trees, thereby making the majority vote among the resulting trees less variable and hence more reliable.
The main difference between bagging and random forests is the choice of predictor subset size m: for instance, if a random forest is built using $m = d$, then this amounts simply to bagging.
In the following a pseudocode for the Random forests algorithm is shown.

Let $\mathcal{D} = \{(x_1, y_1, \ldots, x_n, y_n)\}$ denote the training data, with $x_i = (x_{i,1}, \ldots, x_{i,d})^T$. For $j = 1$ to J:

1. Take a bootstrap sample $\mathcal{D}_j$ of size n from $\mathcal{D}$.
2. Using the bootstrap sample $\mathcal{D}_j$ as the training data, fit a tree using binary recursive partitioning:
   (a) Start with all observations in a single node.
   (b) Repeat the followuing steps recursively for each unsplit node until the stopping criterion is met:
      i. Select m predictors at random from the d availabale predictors.
      ii. Find the best binary split among all binary splits on the m predictors from step above.
      iii. Split the node into two descendant nodes using the split from previous step.

To make a prediction at a new point x, use a majority vote:

$$\hat{f}(x) = \arg\max_y \sum_{j=1}^{J} \mathcal{I}(\hat{h}_j(x) = y)$$

where $\hat{h}_j(x)$ is the prediction of the response variable at x using the j-th tree.

2.5 Boosting

This section is developed with reference to [3] and [20].Boosting is another (with respect to the bagging) algorithmic approach for improving the predictions resulting from a decision tree.

Recall that bagging involves creating multiple copies of the original training data set using the bootstrap method, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model. Notably, each tree is built on a bootstrap data set , independent on the the other trees. Boosting works in a similar way, except that the trees are grown sequentially:each tree is grown using information from previously grown trees. Boosting does not involve bootstrap sampling; instead each tree is fit on a modified version of the original dataset.
This approach uses a generalization of linear predictors to address two specific issues: the bias-complexity trade off and the computational complexity of learning.
We can say that a boosting algorithm amplifies the accuracy of weak learners. Intuitively, a weak learner can be thought of as an algorithm that uses a simple "rule of thumb" to output a hypothesis that comes from an easy-to-learn hypothesis class and performs just slightly better than a random guess. When a weak learner can be implemented efficiently, boosting provides a tool for aggregating such weak hypotheses to approximate gradually good predictors for larger, and harder to learn, classes. It is thus useful to give the following definition:

Definition 2.4. (γ-Weak-Learnability)

- A learning algorithm A is 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 $\delta \in (0,1)$, foe every distribution $\mathcal{D}$ over $\mathcal{X}$, and for every labeling function $f : \mathcal{X} \to \{\pm 1\}$, if the realizable assumption holds with respect to $\mathcal{H}$, $\mathcal{D}$, f, then when running the learning algorithm on $m_{\mathcal{H}}(\delta)$ i.i.d. samples from $\mathcal{D}$ and labeled by f, the algorithms returns a hypothesis h such that, with probability of at least $1-\delta$, $L_{(\mathcal{D},f)}(h) \leq 1/2 - \gamma$.
- A hypothesis class $\mathcal{H}$ is γ-weak-learnable if there exists a γ-weak-learner for that class.

In this section we present two boosting models: AdaBoost (which stands for Adaptive Boosting) and XGBoost (which stands for Extreme Gradient

Boosting).

2.5.1 AdaBoost

The AdaBoost algorithm has access to a weak learner and outputs a low empirical risk hypothesis that is a linear combination of simple hypotheses: we mean that it relies on the family of hypothesis classes obtained by composing a linear predictor on top of simple classes.
In more formal terms, the AdaBoost algorithm receives as input a training set of samples $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ where for each i, $y_i = f(x_i)$ for some labeling function f. The boosting process proceeds in a sequence of consecutive rounds. At round t, the booster first defines a distribution over the samples in S, denoted $D^{(t)}$, that is it is such that $D^{(t)} \in \mathbb{R}_+^m$ and $\sum_{i=1}^{m} D_i^{(t)} = 1$. Then, the booster passes the distribution $D^{(t)}$ and the sample S to the weak learner, in such a way that the weak learner can construct i.i.d. samples according to $D^{(t)}$ and f. The weak learner is assumed to return a "weak" hypothesis h_t, whose error

$$\epsilon_t \doteq L_{D^{(t)}}(h_t) \doteq \sum_{i=1}^{m} D_i^{(t)} \mathcal{I}_{[h_t(x_i) \neq y_i]}, \tag{2.5}$$

is at most $1/2 - \gamma$ (obviously, there is a probability of at most δ that the weak learner fails). Then, AdaBoost assigns a weight for h_t as follows: $w_t = \frac{1}{2}\log(\frac{1}{\epsilon_t} - 1)$. We can observe that the weight of h_t is inversely proportional to the error of h_t. At the end of the round, AdaBoost updates the distribution so that the samples on which h_t errs will get a higher probability mass while samples on which h_t is correct will get a lower probability mass: intuitively, this will force the weak learner to focus on the problematic samples in the next round. The output of AdaBoost algorithm is a "strong" classifier that is based on a weighted sum of all the weak hypotheses.
The pseudocode of AdaBoost is presented in the following:

input:
training set $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$
weak learner WL
number of rounds T
initialize $D^{(1)} = (\frac{1}{m}, \ldots, \frac{1}{m})$.
for $t = 1, \ldots, T$:
invoke weak learner $h_t = WL(D^{(t)}, S)$
compute $\epsilon_t = \sum_{i=1}^{m} D_i^{(t)} \mathcal{I}_{[h_t(x_i) \neq y_i]}$
let $w_t = \frac{1}{2} \log(\frac{1}{\epsilon_t} - 1)$
update $D_i^{(t+1)} = \frac{D_i^{(t)} \exp(-w_t y_i h_t(x_i))}{\sum_{j=1}^{m} D_j^{(t)} \exp(-w_t y_j h_t(x_j))}$ for all $i = 1, \ldots, m$
output: the hypothesis $h_S(x) = \text{sign}(\sum_{t=1}^{T} w_t h_t(x))$.

2.5.2 XGBoost

XGBoost is an abbreviation of eXtreme Gradient Boosting. One of the evident advantages of XGBoost is its scalability and faster model exploration due to the parallel and distributed computing [4].In order to understand XGBoost's algorithm, some basic introduction to how tree boosting methods works will be presented.
Let m be the number of samples in the dataset with p features $S = \{(x_i, y_i)\}_{i=1}^{m}$ ($|S| = m$, $x_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}$). To predict the output, M additive labeling functions are being used

$$\phi(x_i) = \sum_{k=1}^{M} f_k(x_i), \qquad f_k \in \mathcal{F}, \qquad \mathcal{F} = \{f(x) = w_{q(x)}\}, \tag{2.6}$$

where $\mathcal{F}$ is the classification trees' space, q represents the structure of each tree that maps an example to the corresponding leaf index, that is $q : \mathbb{R}^d \to T$, $w \in \mathbb{R}^M$. Further, T is the number of leaves in the tree, each f_k corresponds to an independent tree structure q and leaf weights w, which can also be viewed as a score for ith leaf, w_i. Learning is being executed by

minimization of the following regularized objective:

$$\mathcal{L}(\phi) = \sum_{i=1}^{m} l(y_i, \phi(x_i)) + \sum_{k=1}^{K} \Omega(f_k), \tag{2.7}$$

where $\Omega(f_k)$ is defined as follows

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2. \tag{2.8}$$

Here l is a differentiable convex loss function that measures the difference between the prediction $\phi(x_i)$ and the target y_i. Instead the second term Ω penalizes the complexity of the model by the parameter γ, which penalizes the number of leaves, and by the parameter λ, which penalizes the leaf weights and helps to avoid overfitting. Intuitively, the regularized object will tend to select a model employing simple and predictive functions.
The tree ensemble model in Eq. 2.7 includes functions as parameters and cannot be optimized using traditional optimization methods in euclidean space. Instead the model is trained in an additive manner. formally, let $\phi(x_i)^{(t)}$ be the prediction of the ith observation at the tth iteration, then we will need to add f_t to minimize the following objective

$$\mathcal{L}^{(t)} = \sum_{i=1}^{m} l(y_i, \phi(x_i)^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{2.9}$$

where f_t is chosen greedily so that it improves the model the most. Second-order approximation can be used to quickly optimize the objective in the general setting

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{m} [l(y_i, \phi(x_i)^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{2.10}$$

where $g_i = \partial_{\phi(x_i)^{(t-1)}} l(y_i, \phi(x_i)^{(t-1)})$ and $h_i = \partial^2_{\phi(x_i)^{(t-1)}} l(y_i, \phi(x_i)^{(t-1)})$ are the second order gradient statistics on the loss function. We can remove the constant terms to obtain the following simplified objective at step t

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{m} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2. \tag{2.11}$$

Define $I_j = \{i|q(x_i) = j\}$ as the instance set of leaf j. We can rewrite the equation above as follows

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{m}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2] + \gamma T. \tag{2.12}$$

Now, the expression for the optimal weight w_j^* can be derived from 2.12

$$w_j^* = -\frac{\sum_{i\in I_j} g_i}{\sum_{i\in I_j} h_i + \lambda}. \tag{2.13}$$

Thus, the optimal value is given by

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j}^{T}\frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T. \tag{2.14}$$

Eq. 2.14 can be used as a scoring function to measure the quality of a tree structure q. This score is like the impurity score for evaluating decision trees, except that it is derived for a wider range of objective functions.
Normally it is impossible to enumerate all the possible tree structures q. A greedy algorithm that starts from a single leaf and iteratively adds branches to the tree is used instead. Assume that I_L and I_R are the instance sets of left and right nodes after the split. Letting $I = I_L \cup I_R$, then the loss reduction after the split is given by

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma. \tag{2.15}$$

This formula is usually used in practice for evaluating the split candidates.

2.6 Support Vector Machines

This section is developed with reference to [1]. The Support Vector Machine paradigm is a very useful machine learning tool for learning linear predictors in high dimensional feature spaces. The high dimensionality of

the feature space addresses both sample complexity and computational complexity challenges. The former challenge is takcled by searching for "large margin" separators, that is we mean the following concept: a halfspace separates a training set with a large margin if all the examples are not only on the correct side of the separating hyperplane but also far away from it. Restricting the algorithm to output a large margin separator can yield a small sample complexity even if the dimensionality of the feature space is high. The latter challenge instead is tackled exploiting the idea of kernels.

2.6.1 Margin and Hard-SVM

Let $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ be a training set of examples, where each $x_i \in \mathbb{R}^d$ and $y_i \in \{\pm 1\}$. The training set is said linearly separable if there exists a halfspace, (w, b), such that $y_i = \text{sign}(<w, x_i> + b)$ for all i. An alternative way to write this condition is the following

$$\forall i = 1, \ldots, m \quad y_i(<w, x_i> + b) > 0. \tag{2.16}$$

All halfspaces (w, b) that satisfy this condition are ERM hypotheses; for any separable training sample, there are many ERM halfspaces: the margin is the key idea to understand how the learner chooses which of them to pick.
The margin of a hyperplane with respect to a training set is defined to be the minimal distance betweeen a point in the training set and the hyperplane. If a hyperplane has a large margin, then it will still separate the training set even if we slightly perturb each instance.
Hard-SVM is the learning rule in which we return an ERM hyperplane that separates the training set with the largest possible margin. To formally define Hard-SVM, we first state and demonstrate the following claim:

Proposition 2.5. *The distance between a point x and the hyperplane defined by (w, b) where $||w|| = 1$ is $|<w, x> + b|$.*

Proof. *We can define the distance between a point x and the hyperplane as*

$$\min\{||x - v|| : <w, v> + b = 0\}.$$

Taking $v = x - (<w, x> + b)$ *we have that*

$$<w, x> + b = <w, x> - (<w, x> + b)||w||^2 + b = 0,$$

and

$$||x - v|| = |<w, x> + b|||w|| = |<w, x> + b|.$$

Hence, the distance is at most $|<w, x> + b|$*. Now, taking any other point* u *on the hyperprlane, thus a point such that* $<w, u> + b = 0$*, we have that*

$$\begin{aligned} ||x - u||^2 &= ||x - v + v - u||^2 \\ &= ||x - v||^2 + ||v - u||^2 + 2<x - v, v - u> \\ &\geq ||x - v||^2 + 2<x - v, v - u> \\ &= ||x - v||^2 + 2(<w, x> + b)<w, v - u> \\ &= ||x - v||^2, \end{aligned}$$

where the last equality is beacuse $<w, v> = <w, u> = -b$*. Hence, the distance between* x *and* u *is at least the distance between* x *and* v*, which concludes our proof.* □

On the basis of the preceding proposition, we can conclude that the closest point in the training set to the separating hyperplane is given by $\min_{i=1,\ldots,m}|<w, x_i> + b|$. Therefore, the Hard-SVM rule is

$$\text{argmax}_{(w,b):||w||=1} \min_{i=1,\ldots,m} |<w, x_i> + b| \quad s.t. \quad \forall i,\ y_i(<w, x_i> + b) > 0.$$

We add the fact that whenever we are in the separable case, we can write an equivalent problem:

$$\text{argmax}_{(w,b):||w||=1} \min_{i=1,\ldots,m} y_i(<w, x_i> + b). \tag{2.17}$$

Furthermore, we can give another equivalent formulation of the Hard-SVM rule as a quadratic optimization problem:

$$(w_0, b_0) = \text{argmin}_{(w,b)} ||w||^2 \ s.t.\ \forall i,\ y_i(<w, x_i> + b) \geq 1 \tag{2.18}$$

where the ouput is represented by $\hat{w} = \frac{w_0}{||w_0||}$ and $\hat{b} = \frac{b_0}{||b_0||}$. At this point we can show that the output of Hard-SVM is indeed the separating hyperplane with largest margin. Intuitively, the algorithm searches for w of minimal norm among all the vectors that separate the data and for which $|<w, x_i> + b| \geq 1$, $\forall i$. In other words, finding the largest margin halfspace boils down to finding w whose norm is minimal.Thus it holds that:

Lemma 2.6. *The output of Hard-SVM is a solution of the optimization problem (2.17).*

Proof. *Let (w^*, b^*) be a solution of the optimization problem (2.17) and define the margin achieved by (w^*, b^*) to be $\gamma^* = \min_{i=1,\dots,m} y_i(<w^*, x_i> + b^*)$. Therefore, for all i we have that*

$$y_i(<w^*, x_i> + b^*) \geq \gamma^*$$

which is equivalent to

$$y_i\left(<\frac{w^*}{\gamma^*}, x_i> + \frac{b^*}{\gamma^*}\right) \geq 1.$$

Hence, the pair $\left(\frac{w^*}{\gamma^*}, \frac{b^*}{\gamma^*}\right)$ satisfies the conditions of the quadratic optimization problem (2.18). Therefore, $||w_0|| \leq \left|\frac{w^*}{\gamma^*}\right| = \frac{1}{\gamma^*}$. It follows that for all i,

$$y_i(<\hat{w}, x_i> + \hat{b}) = \frac{1}{||w_0||} y_i(<w_0, x_i> + b_0) \geq \frac{1}{||w_0||} \geq \gamma^*.$$

Finally, knowing that $||\hat{w}|| = 1$ we obtain that $(\hat{w}, \hat{b})$ is an optimal solution of Eq.(2.17). □

2.6.2 Soft-SVM and Norm Regularization

The strong assumption that is made by the Hard-SVM formulation is that the training set is linearly separable: Soft-SVM can be viewed as a relaxation of the Hard-SVM rule that can be applied even if the training set is not linearly separable.

Since the optimization problem in (2.18) enforces the hard constraints $y_i(<w, x_i>+ b) \geq 1$, $\forall i$, a natural relaxation is to allow the constraint to be violated for some of the examples in the training dataset. We can model this idea by introducing nonnegative slack variables, $\xi_1, \ldots, \xi_m$, and replacing each constraint $y_i(<w, x_i> + b) \geq 1$ by the constraint $y_i(<w, x_i> + b) \geq 1 - \xi_i$. The meaning of ξ_i is the measurement of how much the constraint $y_i(<w, x_i> + b) \geq 1$ is being violated. Soft-SVM jointly minimizes the norm of w (representing the margin) and the average of ξ_i (representing the violations of the constraints). The trade off between the two terms is controlled by a parameter λ, so that the overall optimization problem, whose output is always the pair (w, b), has the following form:

$$\min_{w,b,\xi} \left(\lambda ||w||^2 + \frac{1}{m} \sum_{i=1}^{m} \xi_i \right) \quad s.t. \quad \forall i,\ y_i(<w, x_i> + b) \geq 1 - \xi_i,\ \xi_i \geq 0. \tag{2.19}$$

2.6.3 Duality

For simplicity we consider homogeneous halfspaces, that is halfspaces that pass through the origin and are thus defined by $\text{sign}(<w, x>)$, where the bias term b is set to zero. Hard-SVM for homogeneous halfspaces amounts to solving the following problem:

$$\min_{w} ||w||^2 \quad s.t. \quad \forall i,\ y_i <w, x_i> \geq 1. \tag{2.20}$$

Note that we can reduce the problem of learning nonhomogeneous halfspaces to the problem of learning homogeneous halfspaces by adding one more feature to each instance of x_i, thus increasing the dimension to $d + 1$.
Now we want to derive the dual problem of Eq. (2.20) to prepare for the discussion of the kernel trick concept.
We start by rewriting the problem in an equivalent form, considering first the function

$$g(w) = \max_{\alpha \in \mathbb{R}^m : \alpha \geq 0} \sum_{i=1}^{m} \alpha_i (1 - y_i <w, x_i>) = \begin{cases} 0 & \text{if} \quad \forall i,\ y_i <w, x_i> \geq 1 \\ \infty & \text{otherwise} \end{cases}$$

and then rewriting Eq.(2.20) as

$$\min_{w} (||w||^2 + g(w)). \tag{2.21}$$

Rearranging the preceding we obtain another formulation of Eq.(2.20)

$$\min_{w} \max_{\alpha \in \mathbb{R}^m : \alpha \geq 0} \left(\frac{1}{2}||w||^2 + \sum_{i=1}^{m} \alpha_i (1 - y_i <w, x_i>) \right). \tag{2.22}$$

Supposing we flip the order of min and max and knowing that this can only decrease the objective value, we have that

$$\begin{aligned} &\min_{w} \max_{\alpha \in \mathbb{R}^m : \alpha \geq 0} \left(\frac{1}{2}||w||^2 + \sum_{i=1}^{m} \alpha_i (1 - y_i <w, x_i>) \right) \\ &\geq \max_{\alpha \in \mathbb{R}^m : \alpha \geq 0} \min_{w} \left(\frac{1}{2}||w||^2 + \sum_{i=1}^{m} \alpha_i (1 - y_i <w, x_i>) \right). \end{aligned}$$

This inequality is called weak duality and it holds that in this specific case, since the problem is convex and Slater's conditions hold, the strong duality also holds, thus the inequality holds with equality. Therefore the dual problem is

$$\max_{\alpha \in \mathbb{R}^m : \alpha \geq 0} \min_{w} \left(\frac{1}{2}||w||^2 + \sum_{i=1}^{m} \alpha_i (1 - y_i <w, x_i>) \right). \tag{2.23}$$

We notice that we can simplify the dual problem: once α, which is actually a Lagrangian multiplier, is fixed, the optimization problem with respect to w is unconstrained and the objective function is differentiable. Thus, taking the gradient with respect to w of the Lagrangian function, we obtain that at the optimum the gradient equals zero:

$$w - \sum_{i=1}^{m} \alpha_i y_i x_i = 0 \quad \Rightarrow \quad w = \sum_{i=1}^{m} \alpha_i y_i x_i.$$

This show us that the solution must be in the linear span of the examples and this fact is the key concept for the kernel trick.

Finally, plugging the previous result into Eq.(2.23) we can rewrite the dual problem as

$$\max_{\alpha \in \mathbb{R}^m : \alpha \geq 0} \left(\frac{1}{2} \left|\left| \sum_{i=1}^{m} \alpha_i y_i x_i \right|\right|^2 + \sum_{i=1}^{m} \alpha_i \left(1 - y_i < \sum_{j=1}^{m} \alpha_j y_j x_j, x_i > \right) \right). \tag{2.24}$$

Rearranging yields the dual problem

$$\max_{\alpha \in \mathbb{R}^m : \alpha \geq 0} \left(\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j <x_j, x_i> \right). \tag{2.25}$$

Note that the dual problem and its solution as well only involve inner products between instances and does not require direct access to specific elements within an instance. This property is essential to introduce kernels for SVM.

2.6.4 SVM with Kernels

The support vector classifier described so far finds linear boundaries in the input feature space. Although we can make the procedure more flexible by enlarging the feature space using basis expansions such as polynomials or splines. Generally linear boundaries in the enlarged space achieve better training-class separation and translate nonlinear boundaries in the original space. Once the basis functions $h_k(x)$, $k = 1, \ldots, K$ are selected, the procedure is always the same: we fit the SVM classifier using input features $h(x_i) = (h_1(x_i), \ldots, h_k(x_i))$, for $i = 1, \ldots, m$, and produce the (nonlinear) function $\hat{f}(x) = <w, h(x)> + b$.
Recalling the fact that the optimization problem (2.25) and its solution only involve the input features via inner products, we can directly enter the transformed feature vectors $h(x_i)$ in turn of x_i in the problem formulation, obtaining a generalization of the previous inner product. Thus the objective function of (2.25) now becomes

$$\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j <h(x_j), h(x_i)>. \tag{2.26}$$

the advantage of this trick is that, since $h(x_i)$ is involved only through inner products, we need not specify the transformation $h(x_i)$ at all, but require only knowledge of the kernel function

$$K(x_i, x_j) = <h(x_i), h(x_j)>$$

that computes inner products in the transformed space. We can think of a kernel as a function that quantifies the similarity of two observations. Three popular choices for K in the SVM literature are

- dth-Degree polynomial : $K(x_i, x_j) = (1 + <x_i, x_j>)^d$
- Radial basis: $K(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2)$
- Neural Network: $K(x_i, x_j) = \tanh(k_1<x_i, x_j> + k_2)$.

Finally, let us emphasise what the advantage of using kernels is: we have a computational advantage, as using kernels one only need to compute $K(x_i, x_j)$ for all $\binom{m}{2}$ distinct pairs i, j and this can be done without explicitly working in the enlarged feature space, as instead we should do if we simply enlarged the feature space using functions of the original features.

2.7 Convolutional Neural Network

Computer vision is a subfield of deep learning which deals with images on all scales and it allows the computer to process and understand the content of a large number of pictures through an automatic process. the main architecture behind Computer vision is the Convolutional Neural Network, which is a derivative of feedforward neural networks. Its applications are very various such as image classification, object detection, neural style transfer and so on. In particular in this study, as in [21], we want to transform the training data matrix into a grey scale image and then implement a binary classification. More details on this procedure will be given later, for now let us see the theoretical description of CNN, with reference to [22].

2.7.1 Filter processing

The first processing of images was based on filters which allowed, for instance, to get the edges of an object in an image using the combination of vertical-edge and horizontal-edge filters.

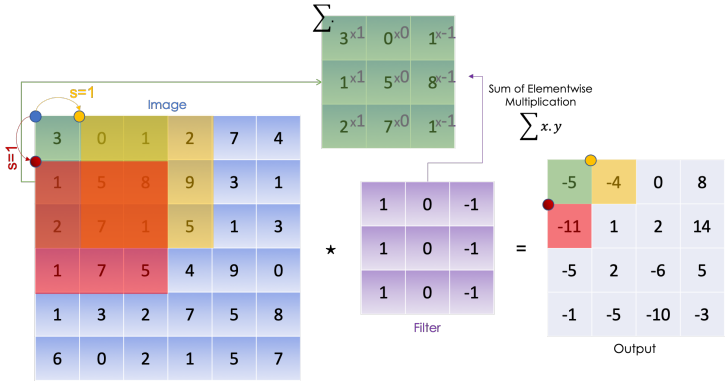

Figure 2.5: We apply a vertical edge filter to a greyscale 6x6 image. We carry out the elementwise multiplication on the first 3x3 block of the matrix and then we consider the following block on the right. We repeat the procedure until we have covered all the potential blocks.

From a mathematical point of view, the vertical edge filter, VEF, is defined as follows:

$$VEF = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = HEF^T$$

where HEF stands for the horizontal edge filter.
To give a simple example, we consider grayscale 6x6 image A, represented by a 2D matrix where the value of each element represents the amount of light in the corresponding pixel. In order to extract the vertical edges from this image, we carry out a convolutional product ($\star$), which is basically the sum of the elementwise product in each block. We carry out the elementwise multiplication on the first 3x3 block of the matrix A and then we consider the following block on the right. We repeat the filter application until we have covered all the potential blocks. A graphic representation of this idea is given in the Figure 2.5.

We can sum up the previous process in the following steps:

$$\texttt{Image} \longrightarrow \texttt{Specific filter} \longrightarrow \texttt{Edges}.$$

Figure 2.6: Diagram of a neural network which takes the image as an input, learns the parameters w_i and outputs a defined target.

Given this example, we can think of using tha same process for any objective where the filter is learned in by neaural network as represented in the diagram in Figure 2.6. The main intuition behind a convolutional neural network is to set a neural network which takesthe image as an input and outputs a defined target to classify it. The parameters w_i are learned using backpropagation technique.

2.7.2 Definition of convolutional neural network

A convolutional neural network is a serie of convolutional and pooling layers which allow extracting the main features from the images responding the best to the final objective, which is in our specific task, binary classification.

Convolution product

Before we explicitly define the convolution product, we first start by defining some basic operations such as the padding and the stride.

Padding

As we have seen in the convolutional product using the vertical edge filter, the pixels on the corner of the image, which is our 2D matrix, are less used than the pixels in the middle of the picture and this means that the information from the edges is thrown away. To solve this problem, we often add a padding around the image in order to take the pixels on the edges into account. In convention, we padde with zeros and denote with p the padding parameter, which represents the number of elements added on each of the

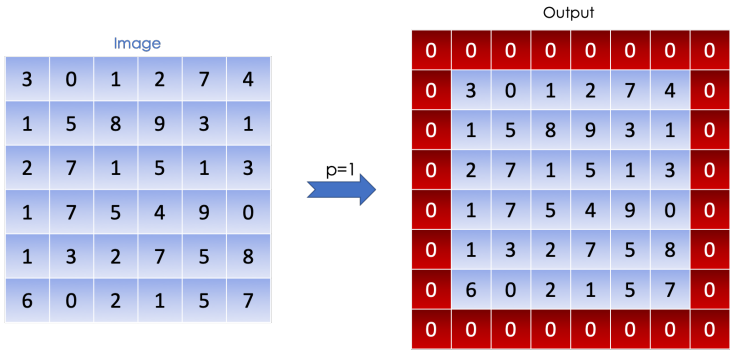

Figure 2.7: We zero pad the image with 1 pixel border.

four sides of the image.

Figure 2.7 illustrates the padding of a greyscale image where $p = 1$.

Stride

The stride is the step taken in the convolutional product and it holds that a large stride allows to shrink the size of the output and vice-versa. We denote with s the stride parameter. For instance, we can observe that in Figure 2.5 the convolutional product is applied with a stride $s = 1$.

Convolution

Once we have defined the stride and the padding we can define the convolution product between a tensor and a filter. So far we have defined the convolution product on a 2D matrix as the sum of the elementwise product; now we can formally define the convolution product on a volume.

An image, in general, can be formally represented as a tensor with the following dimensions:

$$dim(image) = (n_H, n_W, n_C)$$

where n_H is the size of the Height, n_W is the size of the Width and n_C is the number of Channels.

In case of a RGB image, for instance, we have $n_C = 3$, Red, Green and Blue. In convention, we consider the filter K to be squared and to have an odd dimension denoted by f which allows each pixel to be centered in the filter and thus consider all elements around it. Furthermore, when operating the

convolutional product, the filter, namely the kernel K, must have the same number of channels n_C as the image, so that we apply a different filter to each channel. Thus the dimension of the filter is as follows:

$$dim(filter) = (f, f, n_C).$$

From a mathematical point of view, for a given image I and filter K, their convolutional product has the form

$$(I \star K)_{x,y} = \sum_{i=1}^{n_H} \sum_{j=1}^{n_W} \sum_{k=1}^{n_C} K_{i,j,k} I_{x+i-1,y+j-1,k}$$

We also obtain that the filter dimension is as follows:

$$\begin{aligned} dim((I \star K)) &= \left(\left\lfloor \frac{n_H + 2p - f}{s} + 1 \right\rfloor, \left\lfloor \frac{n_W + 2p - f}{s} + 1 \right\rfloor \right); s > 0 \\ &= (n_H + 2p - f, n_W + 2p - f); s = 0 \end{aligned}$$

where $\lfloor x \rfloor$ is the floor function of x. There are some special types of convolution:

- Valid convolution: $p = 0$.
- Same convolution: output size=input size and $p = \frac{f-1}{2}$.
- 1x1 convolution: $f = 1$, it might be useful in some cases to shrink the number of channels n_c without changing the other dimensions (n_H, n_W).

In Figure 2.8 is represented the convolutional product between the image and the filter as a 2D matrix where each element is the sum of the elementwise multiplication of the filter, which is a cube and the submatrix of the given image, which is in turn a subcube.

Pooling

Pooling is the step of downsampling the image's features through summing up the information. This operation is carried out through each channel and thus it only affects the dimensions (n_H, n_W) and keeps n_C intact.

The pooling operation consists of: given an image, we slide a filter without parameters to learn, follwing a certain stride, and we apply a function on the

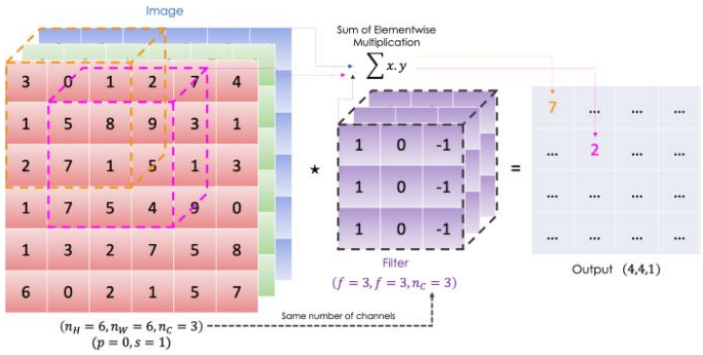

Figure 2.8: The convolutional product between the image and the filter is a 2D matrix where each element is the sum of the elementwise multiplication of the cube (filter) and the subcube of the given image.

selected elements. We have:

$$dim(pooling(image)) = \left(\left\lfloor \frac{n_H + 2p - f}{s} + 1 \right\rfloor, \left\lfloor \frac{n_W + 2p - f}{s} + 1 \right\rfloor, n_C\right); s > 0$$
$$= (n_H + 2p - f, n_W + 2p - f, n_C); s = 0$$

where, in convention, we consider a squared filter with size f and we usually set $f = 2$ and consider $s = 2$. We often apply the following functions on the resulting elements after the filter application:

- Average pooling: we average on the elements present on the filter
- Max pooling: given all the elements in the filter, we return the maximum.

In Fig. 2.9 is illustrated how the average and max pooling work.

Now we explain how to construct a convolutional neural network layer per layer.

One layer of a CNN

The convolutional neural network contains three types of layers:

- Convolutional layer - CONV- followed with an activation function
- Pooling layer -POOL
- Fully connected layer -FC- layer which is basically a layer similar to

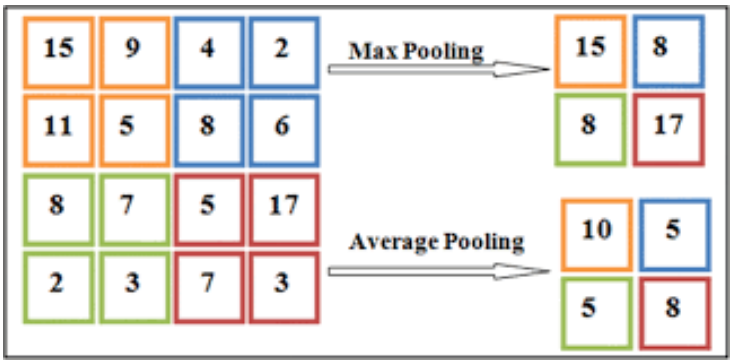

Figure 2.9: We can see that the average pooling takes the average on the elements present on the filter, instead the max pooling takes their maximum.

one from a feedforward neural network.

Convolutional layer

As we have seen before, at the convolutional layer level, we apply convolutional products using different filters, on the input followed by an activation function ψ. In particular, at the l^{th} layer, we denote:

- **Input**: $a^{[l-1]}$ with size $(n_H^{[l-1]}, n_W^{[l-1]}, n_C^{[l-1]})$. Let be $a^{[0]}$ the image in the input
- **Padding**: $p^{[l]}$, **stride**: $s^{[l]}$
- **Number of filters**: $n_C^{[l]}$, where each filter $K^{(n)}$ has the dimension $(f^{[l]}, f^{[l]}, n_C^{[l-1]})$
- **Bias** of the n^{th} convolution: $b_n^{[l]}$
- **Activation function**: $\psi^{[l]}$
- **Output**: $a^{[l]}$ with size $(n_H^{[l]}, n_W^{[l]}, n_C^{[l]})$

Thus the output $a^{[l]}$ for the l^{th} layer is obtained as follows: we apply the non

linear activation function to the sum of the convolutional product between the filter and the input image and the bias term, so we have that

$$\forall n \in [1, 2, \dots, n_C^{[l]}]$$

$$(a^{[l-1]} \star K^{(n)})_{x,y} = \sum_{i=1}^{n_H^{[l-1]}} \sum_{j=1}^{n_W^{[l-1]}} \sum_{k=1}^{n_C^{[l-1]}} K_{i,j,k}^{(n)} a_{x+i-1,y+j-1,k}^{[l-1]} + b_n^{[l]}$$

$$dim(a^{[l-1]} \star K^{(n)}) = (n_H^{[l]}, n_W^{[l]})$$

$$a^{[l]} = [\psi^{[l]}(a^{[l-1]} \star K^{(1)}), \psi^{[l]}(a^{[l-1]} \star K^{(2)}), \dots, \psi^{[l]}(a^{[l-1]} \star K^{(n_C^{[l]})})]$$

$$dim(a^{[l]}) = (n_H^{[l]}, n_W^{[l]}, n_C^{[l]})$$

where

$$n_{H/W}^{[l]} = \left\lfloor \frac{n_{H/W}^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor; s > 0$$
$$= (n_{H/W}^{[l-1]} + 2p^{[l]} - f^{[l]}); s = 0$$
$$n_C^{[l]} = \text{number of filters}$$

So, the learned parameters at the l^{th} layer are the filters, for which we have to learn $(f^{[l]} \times f^{[l]} \times n_C^{[l-1]}) \times n_C^{[l]}$ parameters, and the bias, for which we have to learn $(1 \times 1 \times 1) \times n_C^{[l]}$ broadcasting parameters.
In Fig.2.10 we show a graphic representation of how the convolutional layer works.

Pooling layer
The function of the pooling layer is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. It is important to emphasise that no learning takes place on the pooling layers.
When we consider the l^{th} pooling layer, we refer to the following notation:

- **Input**: $a^{[l-1]}$ with size $(n_H^{[l-1]}, n_W^{[l-1]}, n_C^{[l-1]})$. Let be $a^{[0]}$ the image in the input

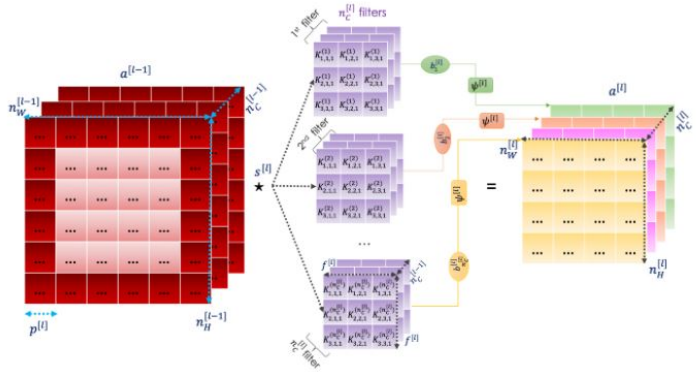

Figure 2.10: Convolutional layer graphic representation.

- **Padding**: $p^{[l]}$, but is rarely used, **stride**: $s^{[l]}$
- **Pooling function**: $\phi^{[l]}$, which can be a max-pooling or an average pooling function
- **Output**: $a^{[l]}$ with size $(n_H^{[l]}, n_W^{[l]}, n_C^{[l]} = n_C^{[l-1]})$, since the pooling layer aims at downsampling the features of the input without impacting the number of the channels.

We can assert that:

$$pool(a^{[l-1]})x, y, z = \phi^{[l]}((a^{[l-1]}_{x+i-1,y+j-1,z})_{(i,j)\in[1,2,\dots,f^{[l]}]^2})dim(a^{[l]}) = (n_H^{[l]}, n_W^{[l]}, n_C^{[l]})$$

with

$$\begin{aligned} n_{H/W}^{[l]} &= \left\lfloor \frac{n_{H/W}^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor; s > 0 \\ &= (n_{H/W}^{[l-1]} + 2p^{[l]} - f^{[l]}); s = 0 \\ n_C^{[l]} &= n_C^{[l-1]}. \end{aligned}$$

In Fig.2.11 we show a graphic representation of how the pooling layer works.

Fully connected layer

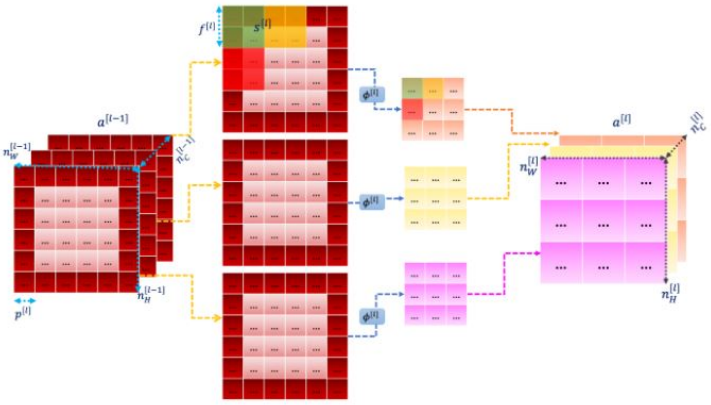

Figure 2.11: Pooling layer graphic representation.

A fully connected layer is a finite number of neurons which takes in input a vector $a^{[i-1]}$ and returns a vector $a^{[i]}$. In general, considering the j^{th} node of the i^{th} layer we have the following equations:

$$z_j^{[i]} = \sum_{l=1}^{n_{i-1}} w_{j,l}^{[i]} a_l^{[i-1]} + b_j^{[i]}$$

$$a_j^{[i]} = \psi^{[i]}(z_j^i)$$

where $w_{j,l}^{[i]}$ are the standard weights of a Multi Layer Perceptron, $b_j^{[i]}$ is the bias term and $\psi^{[i]}$ is the activation function. The input $a^{[i-1]}$ might be the result of a convolutional or a pooling layer with the dimensions $(n_H^{[i-1]}, n_W^{[i-1]}, n_C^{[i-1]})$. Furthermore, in order to be able to plug the input into the fully connected layer we flatten the tensor to a 1D vector having dimension $(n_H^{[i-1]} \times n_W^{[i-1]} \times n_C^{[i-1]}, 1)$, namely $n_{i-1} = n_H^{[i-1]} \times n_W^{[i-1]} \times n_C^{[i-1]}$.
Thus, the learned parameters at the l^{th} layer are:

- **Weights** $w_{j,l}$ with $n_{l-1} \times n_l$ parameters

- **Bias** with n_l parameters.

In Figure 2.12 we sum up the structure of the fully connected layer.

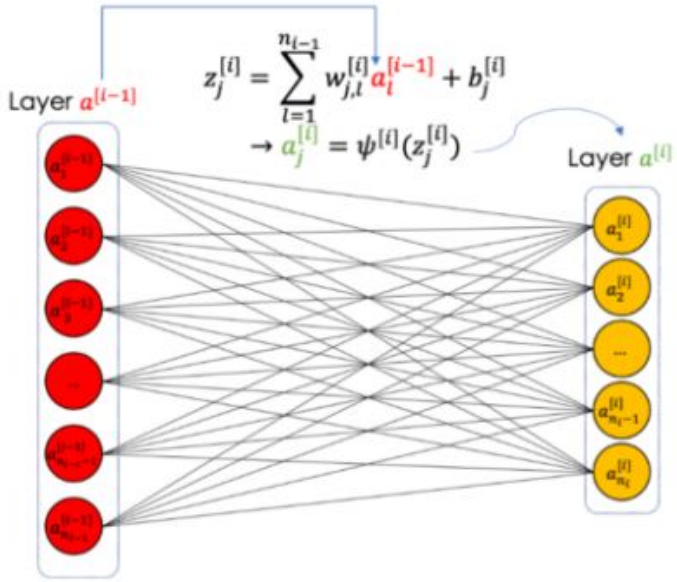

Figure 2.12: Fully connected layer graphic representation.

Finally, to summarise how a CNN works, we can say that in general, a convolutional neural network is a serie of operations: after repeating a serie of convolutions followed by activation function, we apply a pooling fucntion and repeat this process a certain number of time (specified at training time). These operations allow to extract features from the image which will be fed to a neural network described by the fully connected layers, which are regulary followed by activation function as well. The main idea behind the process is to decrease n_H and n_W and increase n_C when going deeper through the network. In Figure 2.13 is shown the synthetic structure of a CNN.

To conclude, convolutional neural networks enable the state of the art results in image processing for two main reasons: the first is parameter sharing property, since a feature detector in the convolutional layer which is useful in one part of the image, might be useful in other ones, and the second is the sparsity of connections property, since in each layer, each output value depends only on a small number of inputs.

In this study, we design a method to transfer every instance of our data set into a pixel matrix, which can be seen as a grey scale image. For each obser-

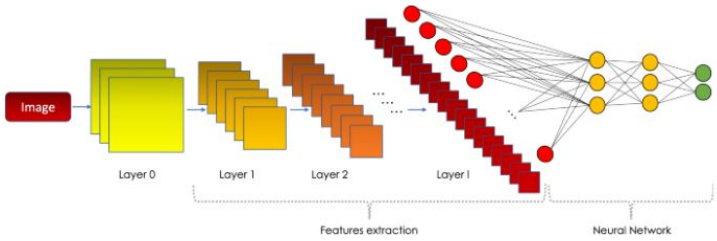

Figure 2.13: CNN structure graphic representation.

vation $x_i = \{x_{i1}, x_{i2}, ..., x_{id}\}$, where d is the number of features in the feature space, we first discretize the continuous variables into categorical ones with k values. Then we reshape every x_ij into a binary value vector $\{l_1, l_2, ..., l_k\}$.

$$l_i = \begin{cases} 1, & \textit{when } x_{ij} \textit{ fall into the } i^{th} \textit{ category} \\ 0, & \textit{otherwise.} \end{cases} \tag{2.27}$$

After the transformation, every observation corresponds to a labeled grey image withe the dimension $k \times d$:

$$y_i + \begin{bmatrix} 0 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 0 \end{bmatrix} \tag{2.28}$$

Thus, the CNN receives the input image and performs the classification task.

2.8 Feature Selection Methods

Often, in a high dimensional dataset, there remain some entirely irrelevant, insignificant and unimportant features. It has been seen that the contribution of these types of features is often less towards predictive modeling as compared to the critical features or even they may have zero contribution. these features cause a number of problems which in turn prevents the process of efficient predictive modeling, for example:

- Unnecessary resource allocation for these features.

- These features act as a noise for which the machine learning model can perform terribly poorly.
- The machine model takes more time to get trained.

Thus the solution is to exploit Feature Selection techniques. Feature Selection in the process of selecting out the most significant features from a given dataset and, in many of the cases, it can enhance the performance of a machine learning model, reducing overfitting and the model's complexity, making it easier to interpret.

There are several types of feature selection methods and they can be classified into: Filter methods, Wrapper methods and Embedded methods. Given the large size of the dataset under study, we have chosen to use a filter method, which by definition does not incorporate a machine learning model in order to determine if a feature is good or bad, unlike wrapper methods, and for this reason is much faster compared to a wrapper method as it does not involve training a model. Among the different filter methods we chose to use the SelectKBest method.

SelectKBest selects features according to the k highest scores obtained from a specific univariate statistical test. Both the value of the parameter k and the statistical test are specified by the user. Due to our classification purposes, as score function we used the Mutual Information.

In general the Mutual Information between two random variables is a non-negative value which measures the dependecy between the variables: it is equal to zero if and only if two random variables are independent, and it holds that higher values mean higher dependency. This property is due to the mathematical definition of the Mutual Information, that involves the Kullback-Leibler divergence. We have that:

Definition 2.7. Let (X, Y) be a pair of random variables with values over the space $\mathcal{X}\times\mathcal{Y}$. If their joint distribution is $\mathbb{P}_{(X,Y)}$ and the marginal distributions are $\mathbb{P}_X$ and $\mathbb{P}_Y$, the Mutual Information is defined as

$$I(X, Y) = D_{KL}(\mathbb{P}_{(X,Y)}||\mathbb{P}_X \cdot \mathbb{P}_Y) \tag{2.29}$$

where D_{KL} is the KullBack-Leibler divergence.

Let $\mathbb{P}$ and $\mathbb{Q}$ be two (discrete) probability distributions defined on the same probability space $\mathcal{X}$. The Kullback-Leibler divergence is a measure of how the first probability distribution is different from the second one and it is defined as the relative entropy of $\mathbb{P}$ with respect to $\mathbb{Q}$:

$$D_{KL}(\mathbb{P}||\mathbb{Q}) = \sum_{x \in \mathcal{X}} \mathbb{P}(x) \log\left(\frac{\mathbb{P}(x)}{\mathbb{Q}(x)}\right) \tag{2.30}$$

or, in other words, it is the expectation of the logarithmic difference between the two probability distributions, also called information gain in machine learning.

2.9 Treatment of imbalanced data with SMOTE

In the data provided, a heavy class imbalance exhibits. An imbalanced data set contains of observations where the classes of the response variable are not approximately equally represented. The imbalance causes a problem when training machine learning algorithms since one of the categories is almost absent, hence poor predictions of new observations of the minority class are expected. In order to increase the performance of the algorithms there are different sampling techniques that can be used. We chose to use SMOTE, which stands for Synthetic Minority Over-sampling Technique. This is an over-sampling approach in which the minority class is over-sampled by creating "synthetic" examples rather than by over-sampling with replacement [5]. In SMOTE algorithm the minority class is over-sampled by taking each minority class sample and introducing synthetic examples along the line segments joining any/all of the k minority class nearest neighbors. Depending upon the amount of over-sampling required, neighbors from the k nearest neighbors are randomly chosen. Synthetic samples are generated in the following way:

1. Take the k nearest neighbors (that belong to the same class) of the considered sample

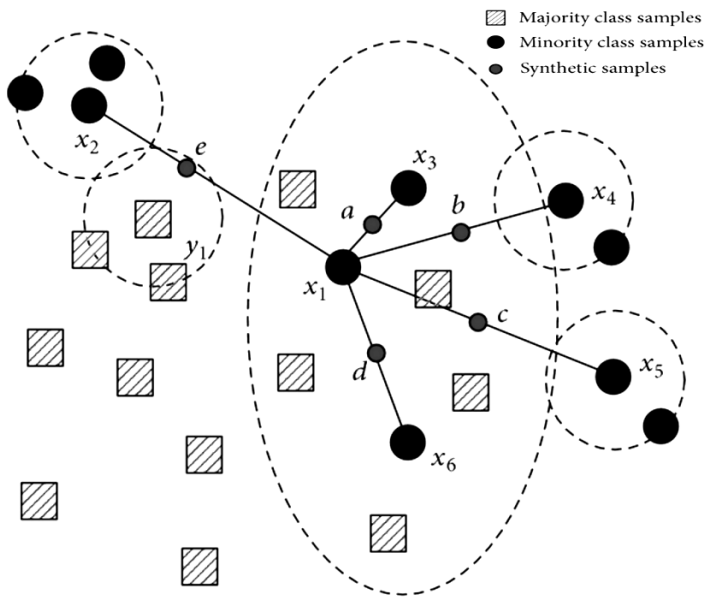

Figure 2.14: Example of a result using SMOTE algorithm.

2. Randomly choose n samples of these k neighbors. The number n is based on the requirement of the over-sampling, for example if 200% is required, then n=2

3. Compute differences between the feature vector of the considered sample and each of the n neighbor feature vectors

4. Multiply each of the differences with a random number between 0 and 1

5. Add these numbers separately to the feature vector of the considered sample in order to create n new synthetic samples

6. Return new synthetic samples.

A graphical example of a result using the SMOTE algorithm is shown in Figure 2.14. The advantage of using the SMOTE algorithm is that the synthetic examples cause the classifier to create larger and less specific decision regions rather than smaller and specific ones. More general regions are thus

learned for the minority class samples rather than those being subsumed by the majority class samples around them. The effect is that the classifier is able to generalize better, so also the model's performances will be better.

2.10 Model evaluation techniques

2.10.1 Confusion Matrix

A common way to evaluate the performance of a classification model with binary responses is to use a confusion matrix. In our project the observed cases of default are defined as positives and non-default as negatives, like in [6]. The possible outcomes are then true positives (TP) if defaulted customers have been predicted to be defaulted by the model. True negatives (TN) if non-default customers have been predicted to be non-default. False positives (FP) if non-default customers hve been predicted to be defaulted, and false negatives (FN) if defaulted customers have been predicted to be non-default. A confusion matrix can be represented as in the Figure 2.15. From

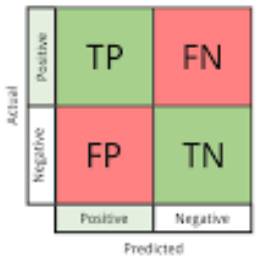

Figure 2.15: Confusion matrix.

a confusion matrix there are certain evaluation metric that can be taken in consideration. The most common metric in a classification task is accuracy, which is defined as the fraction of the total number of correct classifications

and the total number of observations. It is mathematically defined as

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \tag{2.31}$$

The issue with using the accuracy as a metric is when applying it for imbalanced data. In fact if the data contains 99% of one class it is possible to get an accuracy of 99%, if all the predictions are made from the majority class. This can distort our conclusions about the goodness of the model performance and therefore it is not recommended to use this metric in cases of unbalanced data sets.
A metric that is more relevant in the context of this project, where the data set is severely unbalanced, is the specificity. It is defined as

$$Specificity = \frac{TN}{FP + TN}, \tag{2.32}$$

and will be used for explaining the theory behind the receiver operator characteristic curve and its area under the curve in next section.
In terms of business sense, the aim is to achieve a trade-off between loosing money on non-performing customers and opportunity cost caused by declining of a potentially performing customer. Thus, there is a high relevance to analyze how sensitivity and precision metrics are affected by various methods applied, as sensitivity, also named recall, is a measure of how many defaulted customers are captured by the model, while precision relates to the potential opportunity cost. Sensitivity and precision are defined as follows [7] :

$$Sensitivity = \frac{TP}{TP + TN}, \tag{2.33}$$

$$Precision = \frac{TP}{TP + FP}. \tag{2.34}$$

Since sensitivity and precision are of equal importance for our scopes, a trade-off between these metrics is considered. The F-score is the weighted harmonic average of precision and sensitivity. The definition of F-score can

be expressed as

$$F = (1+\beta^2)\frac{Precision \cdot Sensitivity}{Sensitivity + \beta^2 \cdot Precision} = \frac{(1+\beta^2)TP}{(1+\beta^2)TP + \beta^2 FN + FP}, \tag{2.35}$$

where β is a weight parameter. Since in our discussion both measures of precision and sensitivity are equally relevant, the weight is set to $\beta = 1$. Furthermore, since the F-score takes into account both of these measures, we decided that the performance of every method will be primarily evaluated and compared with the regards to this metric.

2.10.2 Receiver Operator Characteristic Curve

A further way to evaluate results from the models is to analyze the receiver Operating Characteristic (ROC) curve and its Area Under the Curve (AUC). In this section we provide the definition of ROC and then we explain the meaninig of AUC.

Let V_0 and V_1 denote two independent random variables with cumulative distribution functions F_0 and F_1, respectively. The random variables V_0 and V_1 describe the outcomes predicted by a model if a customer has defaulted or not. Let c be a cut-off value for the default classification such that if the value from the model is greater or equal to c, a customer is classified as default and non-default otherwise. Further, in this setting, sensitivity and specificity are defined in this following alternative way [8]:

$$Sensitivity(c) = \mathbb{P}(V_1 \geq c) = 1 - F_1(c), \tag{2.36}$$

$$Specificity(c) = \mathbb{P}(V_0 < c) = F_0(c). \tag{2.37}$$

The ROC curve is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. By considering all possible values of the cut-off c, the ROC curve can be constructed as a plot of

sensitivity versus 1−specificity. Let m express $1 - F_0(c)$, then the following definition for the ROC curve is obtained

$$ROC(m) = 1 - F_1\{F_0^{-1}(1-m)\}, \tag{2.38}$$

where $0 \leq m \leq 1$ and $F_0^{-1}(1-m) = \inf\{z : F_0(z) \leq 1-m\}$. The ROC curve is a monotone increasing function mapping $(0,1)$ to $(0,1)$. An uninformative test is one such that $Sensitivity(c) = (1 - Specificity)(c)$ for every threshold c and this situation is represented by ROC curve $ROC(m) = m$, which is a line with unit slope. A perfect test completely separates the defaulted customers and non-default customers, i.e. $Sensitivity(c) = 1$ and $(1 - Specificity)(c) = 1$ for some threshold c. This ROC curve is along the left and upper borders of the positive unit quadrant, as we can see in Fig. 2.16.

The most important numerical index used to describe the behavior of the ROC curve is the area under the ROC curve (AUC), defined by

$$AUC = \int_0^1 ROC(m)dm. \tag{2.39}$$

Referring to Fig. 2.16, line A represents a perfect test with $AUC = 1$, curve B represents a typical ROC curve (for example $AUC = 0.85$), and a diagonal line (line C) corresponding to uninformative test with $AUC = 0.5$. As test accuracy improves, the ROC curve moves toward A, and the AUC approaches 1.

2.11 Cross-validation

Taking up the discussion made on cross validation in the section 1.6 we will now look at this issue in more detail. We know that in order to prevent using the same information in the training phase and the evaluation phase of models, which makes the results less reliable, the data is divided into training set, validation set and test set [9]. We use the training and validation test to find the best model, instead we use the test set only for calculating the

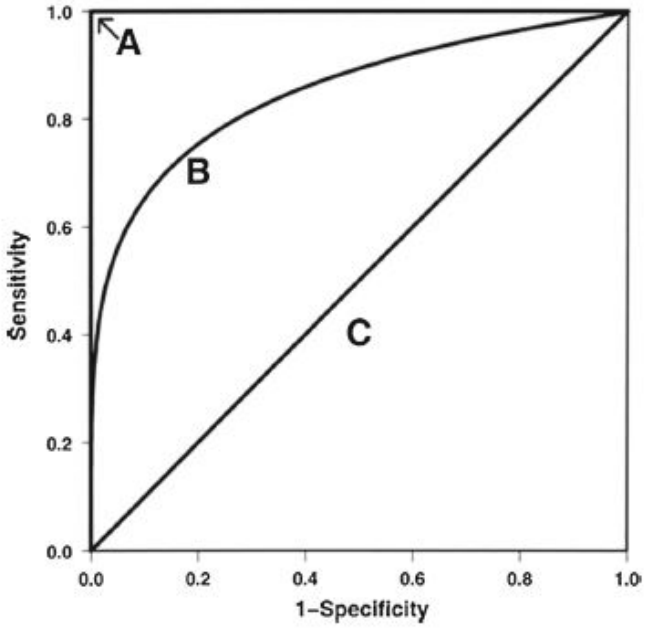

Figure 2.16: Three hypothetical ROC curves representing the accuracy of an ideal test (line A) on the upper and left axes in the unit square, a typical ROC curve (curve B), and a diagonal line corresponding to an uninformative test (line C). As test accuracy improves, the ROC curve moves toward line A.

prediction performance of the best model. The data will therefore be held out until the best model is obtained and indeed this is called the holdout method. The technique for choosing the best model from a set of models is called K-fold cross validation (CV).

K-fold CV involves the procedure where the data set is divided in K roughly equalized sets or folds. One of them is set to be a validation set and the rest are the training set that a model is being fitted on. the procedure is repeated K times and the validation error is being estimated for each time. For example, in Figure 2.17, we have that $K = 5$, where K is a parameter specified by the user, and it means that there have been 5 iterations of the procedure. Let be m the size of the training set and let $\kappa : \{1, \ldots, m\} \rightarrow \{1, \ldots, K\}$ be a mapping function that shows an index of the partition to

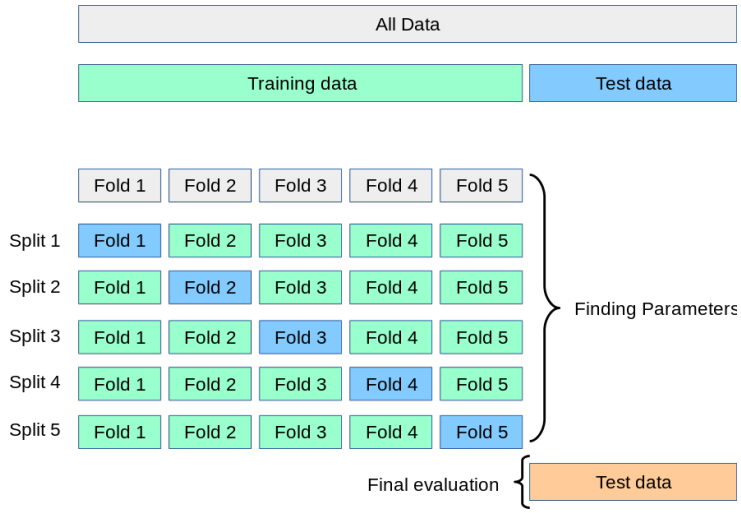

Figure 2.17: K-fold cross validation performed on a given data set.

which observation i is assigned by randomization and $k = 1, 2, \ldots, K$. For the k-th fold, the model is fitted on the $K - 1$ parts and the prediction error is calculated for the k-th part. We denote the fitted model with $\hat{f}^{-k}(x)$ with the k-th part of the data removed, then the CV error is defined as follows:

$$CV = \frac{1}{m} \sum_{i=1}^{m} L(y_i, \hat{f}^{-\kappa(i)}(x_i)), \tag{2.40}$$

where $L(\cdot)$ is the loss function for the respective model. Given a set of competing models $f(x, \alpha)$, with α denoting the index of the model, an exhaustive search for the best α-th model can be performed. Let the α-th fitted model be $\hat{f}^{-k}(x_i, \alpha)$ with the k-th part of the data removed, then the CV error becomes

$$CV(\alpha) = \frac{1}{m} \sum_{i=1}^{m} L(y_i, \hat{f}^{-\kappa(i)}(x_i, \alpha)). \tag{2.41}$$

The objective is to find the α that minimizes the validation error, denoted as $\hat{\alpha}$. This is also known as a hyperparameter search and will be implemented by looping through every combination of a chosen set of hyperparameter values for each machine learning method. When the final model $f(x, \hat{\alpha})$ is obtained, where $\hat{\alpha}$ represents the best combination (that is the combination

that provides the lowest CV error) of hyperparameters, the performance of $f(x, \hat{\alpha})$ will be calculated when predicting on the test set.
In this project, due to the class imbalance problem, K-fold stratified CV has been used in order to preserve proportional representation of the two classes in each fold. In binary classification problem, stratification in cross validation is a technique which rearrange data in a way that each fold contains roughly the same representation of classes between folds.

2.11.1 Cross-validation and SMOTE

When using CV and applying an oversampling technique such as SMOTE we must be careful. We know that in general an oversampling technique duplicates observations of the minority class. However, when we want to use CV to find the best model by applying an oversampling technique to the training data, we have to be very careful about the order in which we implement the procedures. If the oversampling is performed before CV, then the probability of getting duplicate observations in the validation set and the training set is fairly high [11]. The point of the validation set is to calculate the performance of the method on unseen observations. Thus if oversampling is performed before CV there is a possibility that the model has already seen observations and therefore cause biased results. SMOTE is an oversampling technique that does not duplicate the observations but rather synthetically creates new ones. Even if the synthetic new data points generated by SMOTE are not duplicates of an observation, they are still based on an original observation and will therefore cause biased results when predicting on them. So the correct way to proceed is to implement a k fold CV that includes oversampling of the training observations and not the other way around. An example of a correct way of oversampling and use of CV following with an example of an incorrect way is visualized in Figure 2.18.

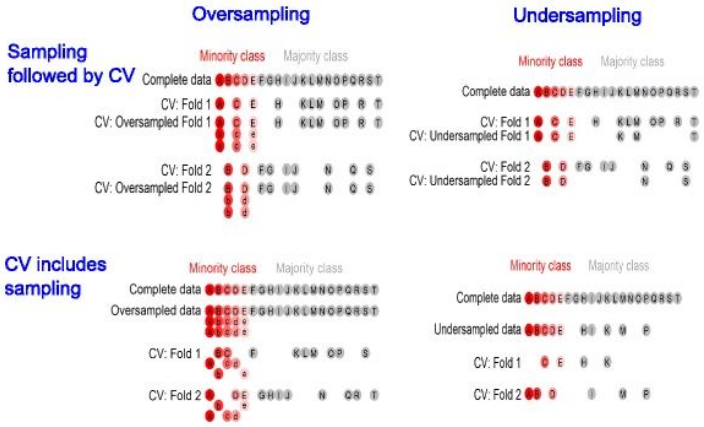

Figure 2.18: Combination of sampling and CV methods. CV includes Sampling (first row) constitutes the correct approach, while Sampling followed by CV (second row) is the incorrect approach. The samples included in the original data set are indicated using upper cases, while their copies are indicated with lower cases.

Chapter 3

Application Example on public data

3.1 Data set description

First of all, we will give an example of applying the previously described models on a public data set. The data set under consideration comes from the Geekbrains AI/Big Data Loan Default Prediction competition, which is a Kaggle InClass competition provided free to academics. Kaggle is an online community of data scientists and Machine Learning enthusiasts. Founded back in 2010, it has become a reference point for anyone approaching this discipline.

The data set studied contains data of customers who have taken out a loan with an unspecified financial institution and based on the characteristics of the customers we want to classify the probability of default (high or low) that these customers have. The data set contains 7'500 observations and 17 columns, of which 16 are the customer features and one is the classification target variable, which takes value 0 if the default probability for a customer is low and value 1 if the default probability is high. More specifically, we will assign label 0 to customers who do not default and label 1 to customers who do default.The table 4.2 shows the names and descriptions of the features.

Variable	**Data type**	**Description**
home ownership	categorical	to check if the borrower owns a house or stays in rent
annual income	numeric	amount of borrower's annual income
years in current job	categorical	how many years the client has been practising the current profession
tax lines	numeric	how many tax liens the client has against him
number of open accounts	numeric	how many accounts the costumer has
years of credit history	numeric	how long all of the borrower's credit accounts have been open
maximum open credit	numeric	amount of the maximum open credit
number of credit problems	numeric	how many problems the client has had during his credit history
months since last delinquent	numeric	how many months have passed since the last delinquency
bankruptcies	numeric	how many times the client has gone bankrupt
purpose	categorical	purpose for which the loan was requested
term	categorical	short- or long- term loan indicator
current loan amount	numeric	borrower's current loan amount
current credit balance	numeric	borrower's current credit balance
monthly debt	numeric	amount of borrower's monthly payment
credit score	numeric	indicator of the customer's creditworthiness, the higher the better
credit default	categorical	target variable indicating if the customer will default or not

Table 3.1: Data dictionary

As can be seen from the 4.2, the data set has four categorical independent variables and one dependent variable, i.e. our target variable, which is also categorical (since our task is to perform a binary classification). We therefore give below the description of the labels of each categorical variable.

- **home ownership** has following labels: have Mortgage, home Mortgage, own Home, rent;

- **years in current job** has the following labels: 10 + years, 9 years, 8 years, 7 years, 6 years, 5 years, 4 years, 3 years, 2 years, 1 year, $<$ 1 year;

- **purpose** has the following labels: debt consolidation, other, home improvements, business loan, buy a car, medical bills, major purchase, take a trip, buy house, small business, wedding, moving, educational expenses, vacation;

- **term** has the following labels: short term, long term.

Since we found that the interpretability of the credit score variable was unclear, as the score scale was outside the usual ranges commonly used by financial institutions for credit scoring (e.g. FICO credit scores, which range from 300 to 850), we decided to ignore it in our further analysis.
The data set also had a certain amount of missing values in the features, the amount of which is shown in table 3.2. We know that the handling of missing values for quantitative and qualitative variables is different. For quantitative variables it is usual to replace the missing values by a standard value, the mean or the median. However, although there are three different options, the most correct one to use is undoubtedly the median, as the mean suffers from the presence of outliers and can affect the goodness of fit of the model. Therefore we have replaced the missing values of the variables annual income, years in current job and months since last delinquent with the median. For the qualitative variables, on the other hand, it is usual to replace the missing values by the mode or by adding a new label to indicate the missing value.

Here, choosing to interpret the bankruptcies variable as a qualitative variable, since it has a finite number of integer values that we can interpret as levels, we decided to replace the missing values with the mode.

Feature name	**Number of null values**
home_ownership	0
annual_income	1557
years_in_current_job	371
tax_liens	0
number_of_open_accounts	0
years_of_credit_history	0
maximum_open_credit	0
number_of_credit_problems	0
months_since_last_delinquent	4081
bankruptcies	14
purpose	0
term	0
current_loan_amount	0
current_credit_balance	0
monthly_debt	0
credit_default	0

Table 3.2: Number of null values for each feature considered in our analysis.

3.2 Exploratory analysis

Let us now proceed with an initial exploratory analysis of the data set. Data visualisation is very useful in the early stages of analysis, helping to explain content by organising data in a more understandable way and highlighting trends and outliers. Effective visualisation helps to expose content by removing the superfluous from the data and bringing useful information to the fore.

First of all, let's see how many default customers are present in the data set, as figure 3.1 shows. Counting the values, we see that there are 5,387 non-defaulting customers and 2,113 defaulting customers, i.e. about 72% of the data set consists of customers who will repay their obligations while about 28% of the data set consists of customers who have defaulted on their payments.

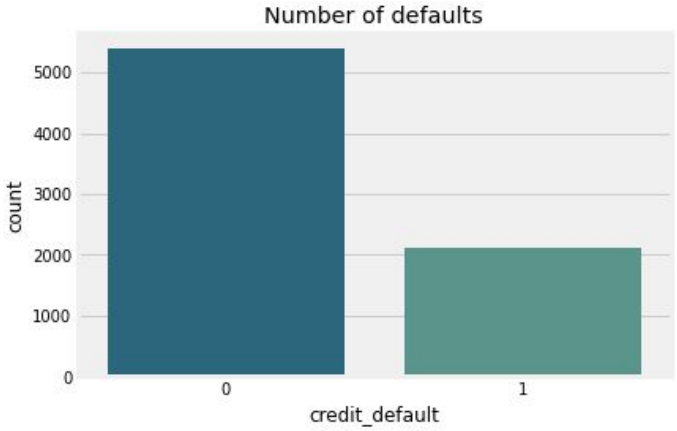

Figure 3.1: Number of default and non-default clients.

Let us now look at the frequencies of the different labels in the categorical variables with respect to the credit default target variable. As we can see from figure 3.2, the most frequent label for non-default customers is having a mortgage on their house or renting, while only a small percentage of them own their own house. Interestingly, the number of defaults is almost the same for home mortgage holders and renters. This trend in the data is quite reasonable as in addition to the payments due to the credit obtained, the customer has to pay the house mortgage or rent and therefore may fall into financial difficulties.

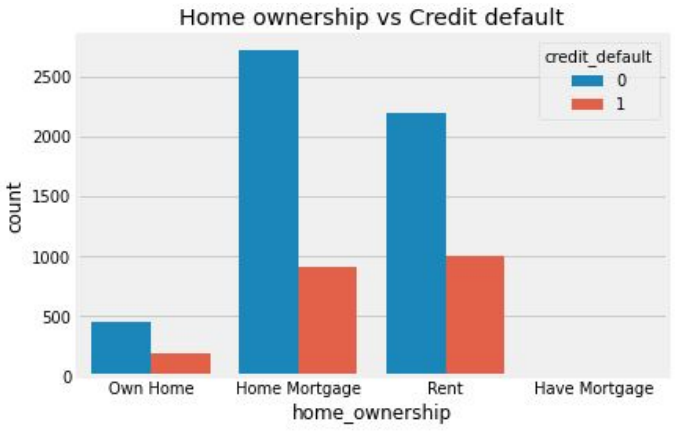

Figure 3.2: Labels of home ownership variable with respect to the target variable.

Looking at the bar plot of the bankruptcies versus credit defaults variable in figure 3.3, we see that the vast majority of the observations had no financial problems in their credit history, and only a small proportion of the clients suffered from serious financial problems. However, within the customers who have never been bankrupt, half are in default and the same is true for those who have been bankrupt.

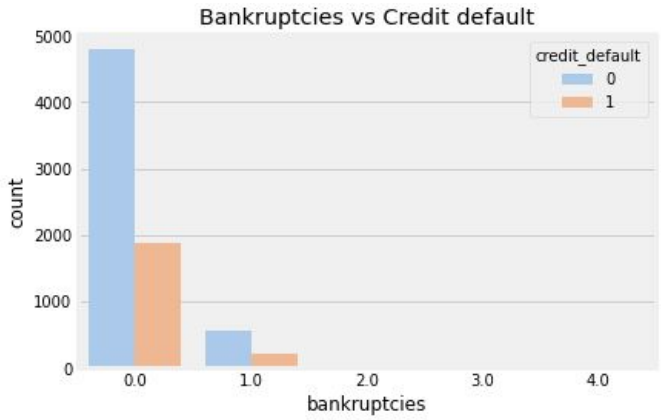

Figure 3.3: Labels of bankruptcies variable with respect to the target variable.

In figure 3.4 we see that the trend observed above for the variable bankrupt-

cies is repeated for the variable number of credit problems when we look at it in relation to the variable credit defaults. Here too, the majority of customers have never had credit problems, although within this set we also find the highest number of customers in default.

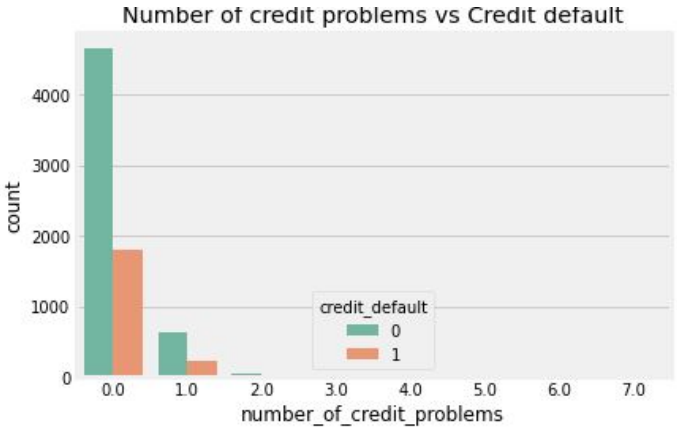

Figure 3.4: Labels of number of credit problems variable with respect to the target variable.

Looking then at the term plotted against the credit default variable in figure 3.5, we see that the number of defaulting customers is very similar in the two groups of customers with a long or short term loan. We could therefore assume that the default of a customer does not depend strongly on whether the loan obtained is short or long term.

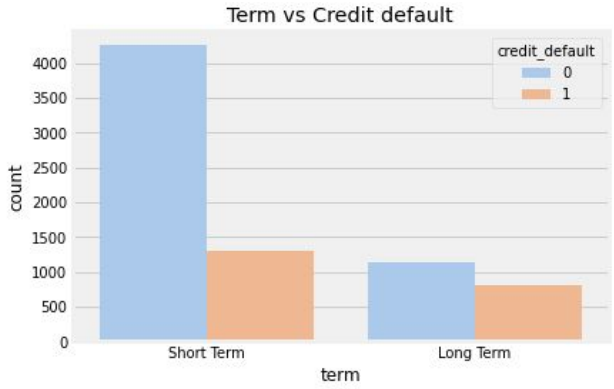

Figure 3.5: Labels of term variable with respect to the target variable.

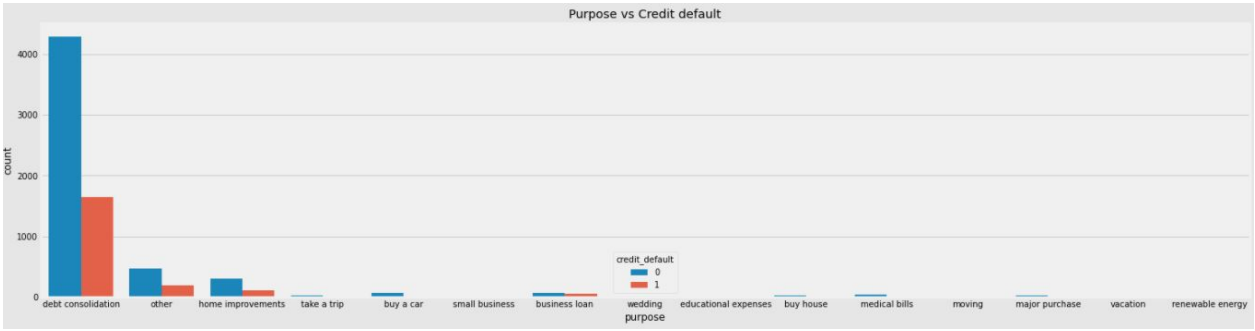

Figure 3.6: Labels of purpose variable with respect to the target variable.

From the figure 3.6 we see that most defaults come from customers who need to consolidate a debt. In second place we find defaults from other purposes than those listed in the figure and finally we find defaults from customers seeking credit for home improvements. We also note that applicants whose purpose of the loan is business lending are almost equally payment compliant and non-payment compliant. We can therefore conclude that the main purpose for borrowing is to consolidate another debt, thus increasing one's personal debt. As can be seen from the graph, this is not always a successful economic strategy, since if you do not have sufficient finances you risk defaulting on at least one of the two debt fronts.
Box plots of the following quantitative variables are shown in figure 3.7: annual income, maximum open credit, current loan amount, current credit balance and monthly debt. We can observe that the data of these variables are rather symmetrical, as the median is in the middle of each box. Annual income is the variable with the narrowest interquartile range, while current credit balance is the one with the widest interquartile range. All the variables considered, as we can see, present a considerable number of outliers. In particular, there is a client who presents a value equal to zero for the variables maximum open credit, current credit balance and monthly debt: it was decided to remove this anomalous observation from the data set.

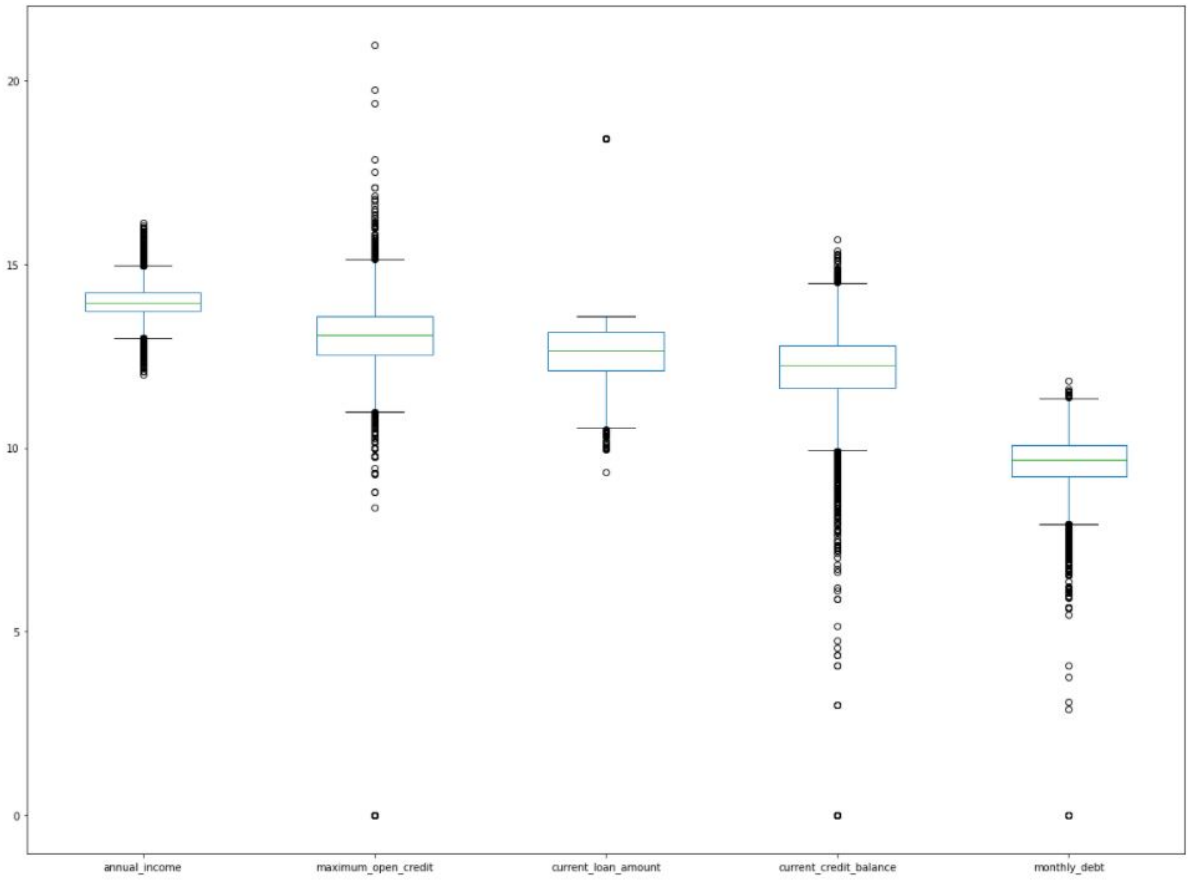

Figure 3.7: Box plot of some quantitative variables.

Finally, we comment on the correlation matrix in figure 3.8. We observe that number of credit problems and tax liens report a correlation of 0.6. This seems to be quite reasonable, as the tax lien is a first degree credit, issued by the county to recover unpaid property taxes and therefore it makes sense that it is strongly correlated with more general credit problems. We find that the bankruptcies variable is also strongly correlated (correlation of 0.73) with the number of credit problems, as a bankruptcy can be counted as a credit problem. With lower positive correlations, but still considerable, we find the variable monthly debt and annual income (correlation 0.58) and the same variable monthly debt and the number of open accounts (correlation 0.41). Finally, the correlation between credit default and credit score is quite

high (0.44). This is more than justified, as the credit score could be the target variable in itself, as it completely reflects the creditworthiness of the client. At this point, the target variable credit default would only be a further redundant encoding of the customer's creditworthiness. For this reason, and because of problems with the interpretability of the credit score variable, we decided to exclude the credit score variable from our analysis.

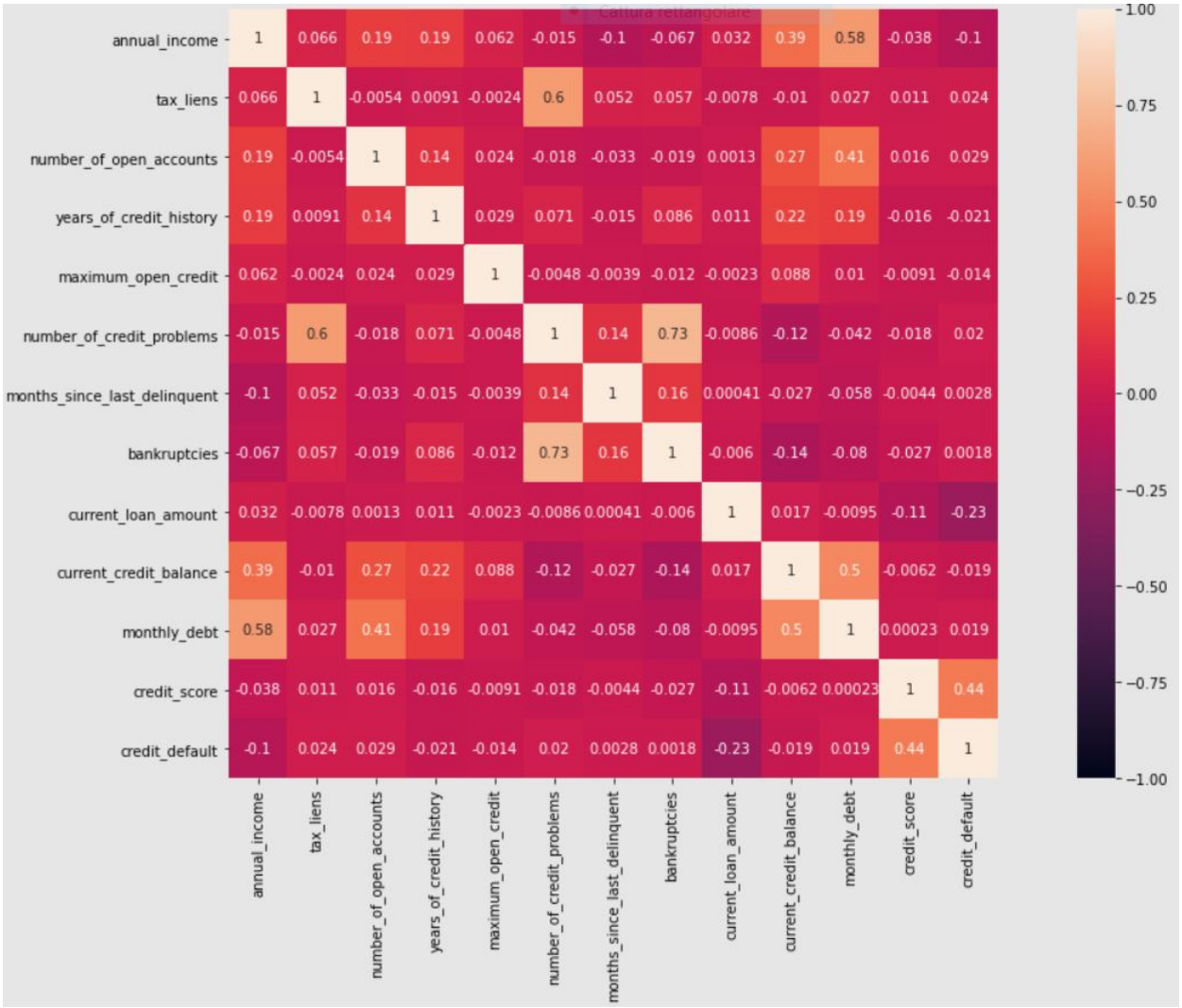

Figure 3.8: Correlation Heatmap.

3.3 Results from models

Below in table 3.3 are the results obtained by training and testing the models on the data set under analysis. The evaluation metrics considered are precision, recall, F-score and Area Under Roc Curve (AUC). We will use the F-score metric as the main metric to decide the best model from the point of view of our econometric analysis. However, we will also consider the AUC as a metric for evaluating the performance of our models, following the common practice of model evaluation in data science.

Model	**Precision**	**Recall**	**F-score**	**AUC**
Decision Trees	0.63	0.64	0.64	0.641
KNN	0.75	0.61	0.62	0.726
CNN	0.70	0.64	0.65	0.735
Random Forest	0.76	0.62	0.63	0.758
Support Vector Classifier	0.88	0.59	0.59	0.692
AdaBoost	0.80	0.64	0.66	0.778
Logistic Regression	0.76	0.61	0.61	0.759
XGBoost	0.82	0.63	0.64	0.776

Table 3.3: Results from models with respect the evaluation metrics.

From the table we can see that the highest Precision results were achieved by the Support Vector Classifier (0.88) and Extreme Gradient Boosting (0.82), while Decision Trees achieved the lowest score (0.63). As far as the Recall scores are concerned, we can see that the highest value reached is 0.64 and this is the same for the Decision Trees, the Convolutional Neural Network and AdaBoost. Remembering that the F-score is the harmonic mean between Precision and Recall and represents our primary metric for evaluating the performance of models from a business point of view, we observe that AdaBoost has the highest F-score of all (0.66) and after it we find the Convolutional Neural Network (0.65). Finally, if we look at the AUC values of the various models under examination, we can see that in first place we

find AdaBoost and in second place we find XGBoost, with respective AUC scores of 0.778 and 0.776. We can therefore conclude that the model with the best out-of-sample performance is AdaBoost, and the optimal parameters of the model are given below:

- *base estimator*: it is the base estimator from which the boosted ensemble is built. In our case it is Decision Tree Classifier.
- *n estimators*: it is the maximum number of estimators at which boosting is terminated. in case of perfect fit, the learning procedure is stopped early. In our case the maximum number of estimators is set to 100.
- *learning rate*: it is the weight applied to each classifier. There is a trade-off between the learning rate and the maximum number of estimators parameters. Our optimal value for the learning rate is equal to the default one, i.e. equal to 1.
- *loss*: it is the loss function to use when updating the weights after each boosting iteration. We set a linear loss.

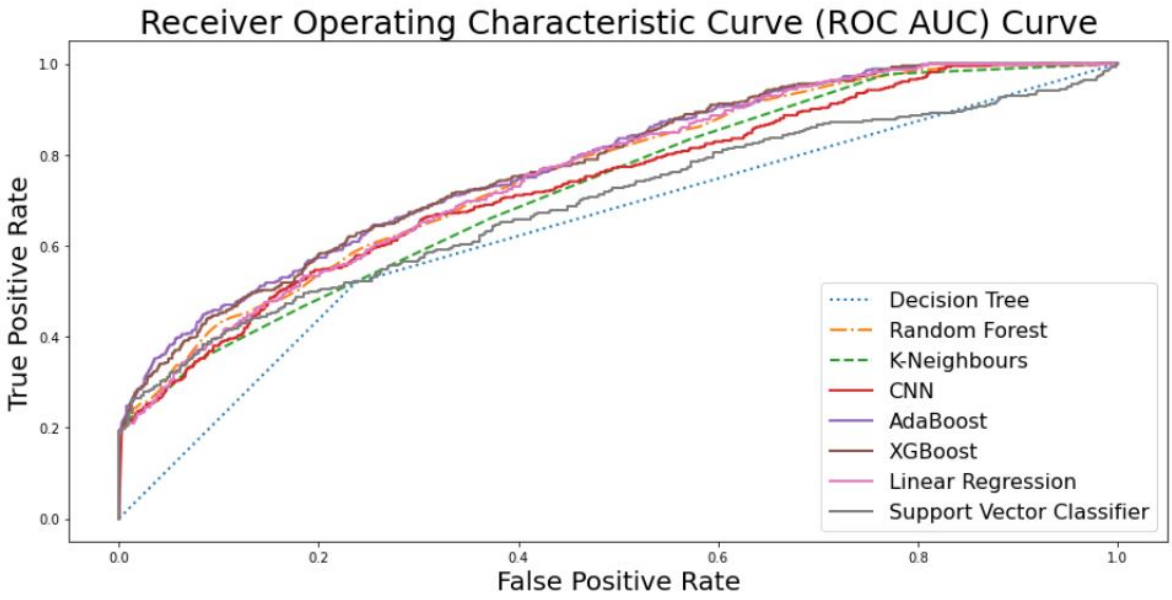

Figure 3.9: ROC curves of models implemented.

In Figure 3.9ROC curves of the analysed classifiers are shown. We know that the ROC curve plots 1 minus specificity (the false positive rate) and sensitivity (true positive rate) when a threshold is varied. We remind that the true positive rate is the fraction of defaulters that are correctly identified, using a give threshold value. The false positive rate instead is the fraction of non-defaulters that we classify incorrectly as defaulters, using the same threshold value. ROC curves are useful for comparing different classifiers, since they take into account all possible thresholds. Also, an ideal ROC curve will hug the top left corner, indicating a high true positive rate and a low false positive rate, so the larger the AUC the better the classifier.
From Figure 3.9 we can see that the Decision Trees, with the lowest AUC score of 0.641, have the worst ROC curve, being furthest from the top left corner. On the other hand, the AdaBoost and XGBoost classifiers are those that report ROC curves that are closest to the upper left corner, reflecting their respective good AUC scores. If we look closely, however, we can see that the ROC curve of AdaBoost, although very close to that of XGBoost, dominates it for every threshold value. This once again confirms that AdaBoost is the model with the greatest predictive power.

Chapter 4

Application Example on real data

4.1 Data set description

In this section we present the data set used in our work, which comes from a private database of a Swiss bank. The data set covers approximately 54.669 private residential mortgages originated between December 31, 2019 and December 31, 2020 and it consists of 32 columns, including the target variable. The features include personal information about the client, such as nationality, age, professional occupation, quantitative information about the amount of the loan and the client's financial situation, and finally information about the property the client intends to purchase through the loan, such as the size in square metres, the condition of the building and others.
The target variable was feature engineered, i.e. in the original data set there was a feature representing the credit score of each customer, so we chose a threshold to discriminate between defaulting and non-defaulting customers. The credit scores were on a descending scale of creditworthiness from 1 to 18, with the default status being the last: we chose to put the default threshold at the tenth credit score, thus assigning the default status to any customer with a credit score greater than or equal to 10. We therefore created the binary target variable, named "Default", which takes value 0 when the customer has a rating lower than 10, i.e. when the customer is financially reliable, while it

takes value 1 when the customer has a rating higher than 10 and is therefore not considered a creditworthy customer.

4.2 Results from models

In this chapter, results obtained from different models will be presented and discussed.
The results were obtained with the data set provided by a private database from a Swiss bank. Three different data sets were studied, where one of them is the data set containing all the features, while the others were obtained by a feature selection technique, called SelectKBest, iterated 2 times, thus containing 16 and 9 variables respectively.
To compensate for the significant data imbalance, a minority class oversampling technique called SMOTE was applied: the minority class was oversampled until its magnitude corresponded to 50% of the majority class.
As mentioned in section 2.10, performance of the methods is evaluated by sensitivity, precision, F-score and AUC-score with the F-score as the primary metric. All of the results are presented in Table 4.2.

The analysis of the results is divided as follows: first, for each dataset obtained, we will comment on the models with the best performance with respect to the metrics considered. Then we will comment on the best model for each metric. Finally, we will give an idea of the scale of goodness of the models and comment, with respect to each metric, on the impact of the feature selection method. With respect to the dataset on which no feature selection method was applied, we can say the following: KNN and CNN achieved the best precision and f-score scores (0.6637 and 0.6867 for CNN and 0.7177 and 0.2537 for KNN), while for recall the highest values were obtained with Logistic Regression (0.6969) and CNN (0.7112). Finally, KNN and XGB obtained the best AUC scores, 0.95 and 0.73 respectively. Considering then the dataset produced by the 16 features selected by the SelectKbest method with K=16, we can say that KNN and CNN reached again the best performances

in precision,0.5555 and 0.6938 respectively, while CNN and Logistic Regression reached the highest F- scores (0.6150 and 0.1654, respectively) and recall (0.5223 and 0.6969, respectively). Finally, CNN and XGB had the highest AUC values of 0.71 and 0.72. Finally, considering the dataset composed of 9 features, produced by the feature selection method SelectKBest setting K=9, we observe that the highest precision value was reached by CNN (0.6938) and second to it are RandomForest (0.1793) and AdaBoost (0.1718). The same applies to the F-score, where the Logisic Regression score (0.1938) is also remarkable. Observing instead the recall scores we can see that the Logistic Regression obtained the highest score of all (0.6521), surpassing also that of the CNN (0.5523). Finally, in terms of AUC score, CNN and the boosting methods AdaBoost and XGBoost are quite close, one being 0.71 and the other two 0.69.

Analysing now the best model with respect to each evaluation metric considered, the following observations can be made. The KNN model on the dataset to which feature selection was not applied reported the best precision value (0.7177), even higher than that reported in general by CNN. As for the F-score, which is our primary metric, the best model is definitely CNN on the original dataset (0.6867). However, in second place after the CNN, we find again the KNN model applied to the whole dataset, whose F-score value is equal to 0.2537 and differs considerably from all the other F-score values of the remaining Machine Learning models. The highest recall value was reached by the CNN applied to the original dataset (0.7112). However, the recall score reached by the Logistic Regression model applied to the whole dataset and to the dataset containing 16 features (it seems that the logistic regression did not benefit in terms of recall from the feature selection) is not very different from this value (0.6969). Finally, in terms of AUC score, the highest score of all is reached by the KNN model applied to the whole dataset (0.95) and in second place after it is the XGBoost model applied both to the whole dataset and to the dataset containing 16 features.

Discussing then the impact of the feature selection method applied to the

data set, from table 4.2 we can observe the following. The feature selection of 16 features had a slight positive impact in terms of mean precision. Feature selection of 9 features had a significant positive impact on the average F-score and recall scores. Finally, the average AUC score, as we can see, did not benefit from the feature selection, as it became lower and lower as the number of features included in the data set decreased.

To conclude, we can say that CNN applied to the entire data set turns out to be the model with the highest predictive power and in second place we have the KNN model applied to the entire data set. These results say a lot about the great predictive power that Deep Learning methods have, in particular the Convolutional Neural Network applied to the unusual setting represented by the prediction of defaults in the credit score analysis. Therefore, although compared to other Machine Learning models whose interpetability is greater, it has been experimentally proven that in our study CNN outperforms the other models considered and therefore could provide considerable support and improvement in the credit score analysis .

Model	Precision	F-score	Recall	AUC
Models without FS				
LogisticRegression	0.0938	0.1654	0.6969	0.63
KNN	0.7177	0.2537	0.1514	0.95
DecisionTrees	0.1476	0.1670	0.1923	0.63
RandomForest	0.1408	0.1336	0.1270	0.65
AdaBoost	0.19	0.1109	0.0783	0.68
XGBoost	0.3035	0.0417	0.0233	0.73
SupportVectorMachine	0.0669	0.0613	0.0566	0.57
CNN	0.6637	0.6867	0.7112	0.72
Model with 16 features				
LogisticRegression	0.0938	0.1654	0.6969	0.63
KNN	0.5555	0.0130	0.0065	0.86
DecisionTrees	0.1050	0.1342	0.1857	0.51
RandomForest	0.1448	0.1541	0.1646	0.67
AdaBoost	0.2184	0.1537	0.1185	0.69
XGBoost	0.4166	0.0935	0.0527	0.72
SupportVectorMachine	0.0974	0.1262	0.1791	0.59
CNN	0.6938	0.6150	0.5223	0.71
Model with 9 features				
LogisticRegression	0.1138	0.1938	0.6521	0.68
KNN	0.1057	0.1635	0.3610	0.60
DecisionTrees	0.0983	0.1656	0.5243	0.57
RandomForest	0.1793	0.1886	0.1989	0.67
AdaBoost	0.1718	0.1997	0.2384	0.69
XGBoost	0.2309	0.1578	0.1198	0.69
SupportVectorMachine	0.1196	0.1715	0.3030	0.61
CNN	0.6938	0.6150	0.5523	0.71

Table 4.1: Final results from the models investigated.

Nbr of features	Average precision	Average F-score	Average recall	Average AUC
23	0.2871	0.202	0.254	0.695
16	0.290	0.181	0.240	0.672
9	0.21	0.2319	0.3687	0.652

Table 4.2: Average results of evaluation metrics considered.

Chapter 5

Economic evaluation of model performance

This section is developed with reference to [23] and it is a theoretical discussion of the economic evaluation of the model performance.
An important question for a credit risk manager is to what extent the statistical performance gains obtained by implementing various machine learning methods have a positive financial impact for the bank. One economic valuation method is to estimate the amount of regulatory capital induced by the estimated probabilities of default. A similar comparison approach has been proposed by [24] for loss-given-default (LGD) models. However, this approach requires the calculation of other Basel risk parameters, in particular LGD and exposure to default (EAD), and therefore requires specific information on consumers and loan conditions, which is not in the scope.
An alternative approach is to compare the costs of misclassification [25]. From the confusion matrix, it can be clearly seen that the number of False Positives represents a Type I statistical error, while the number of False Negatives represents a Type II statistical error. The misclassification cost can be estimated from Type I and Type II errors weighted by their probability of occurrence.
Formally, let C_{FN} be the cost associated with a Type I error (the cost of

granting credit to a bad customer) and C_{FP} be the cost associated with a Type II error (e.g., the cost of rejecting a good customer). Thus, the misclassification error cost (MC) is defined as

$$MC = C_{FP} \cdot FPR + C_{FN} \cdot FNR, \tag{5.1}$$

where FPR is the false positive rate and FNR is the false negative rate. There is no consensus in the literature about how to determine C_{FN} and C_{FP}. Two alternatives have been proposed. The first method fixes these costs by calibration based on previous studies [26]. For example, [27] set C_{FN} to 5 and C_{FP} to 1. In figure 5.1 we give an example of the calculation of the misclassification function of the errors (setting the parameters such that C_{FN} is 5 and C_{FP} is 1) on the public data set provided by the Kaggle platform, with respect to the various models under examination. We can observe how the Decision Tree is the method that presents the lowest cost of misclassification while the Support Vector Classifier is the method whose error is the most expensive of all.

The second method evaluates misclassification costs for different values of C_{FN} to test as many scenarios as possible [28]. Although there is no consensus on how to determine these costs, it is generally acknowledged that the cost of granting credit to a bad customer is higher than the opportunity cost of rejecting a good customer [29],[27].

We could also consider a second measure of performance, namely, the expected maximum profit (EMP) introduced by [30], to compare the models from an economic viewpoint. The EMP takes into account the profits received by the non-defaulters and the losses caused by the defaulters. This allows us to compute an EMP value that is expressed as a percentage of the total loan amount and measures the incremental profit relative to not building a credit scoring model. The EMP is based on the following utility function of the decision maker, which is in our case the bank:

$$P(t; b, c, c^*) = (b - c^*) \cdot \pi_0 \cdot F_0(t) - (c + c^*) \cdot \pi_1 \cdot F_1(t) \tag{5.2}$$

where t is a cutoff; b is the benefit associated with a true positive; c is the cost associated with a false positive; c^* is the cost associated with an individual case; π_0 and π_1 are the prior probabilities of non-default and default, respectively; and $F_0(t)$ and $F_1(t)$ are the corresponding cumulative density functions.The parameters b and c are calibrated using the LGD and return on investment (ROI, see [30] for more details), which is basically the difference between the current value and the cost of the credit provided, divided by the cost of the same credit.

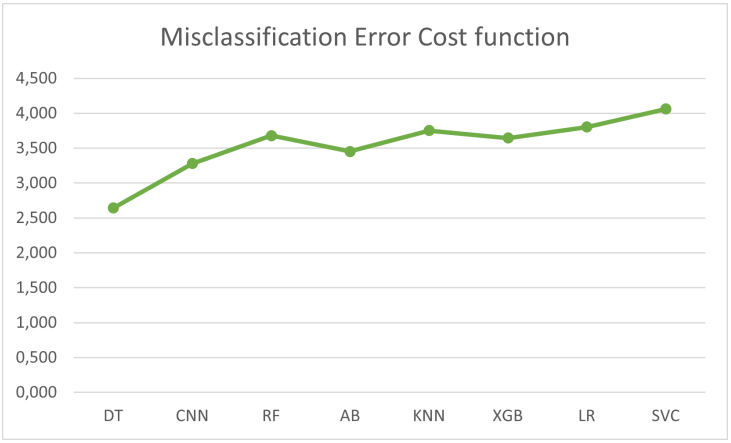

Figure 5.1: Graph of the misclassification function calculated on the data of the public data set under consideration and with respect to the methods considered in the work.

Chapter 6

Conclusions

The research question attaining to answer in this work was: For a chosen set of Machine Learning and Deep Learning techniques, which technique exhibits the best performance in predicting default against a specific model evaluation metric?
For the sake of brevity, but also for reasons of relevance and priority, we will only comment on the results obtained from the models implemented on the private data set. The overall results showed that the Convolutional Neural Network applied to data sets containing all the features under analysis, 16 features and 9 features respectively, performed best against the F-score. However, due to the complexity of the model, the data set containing 9 variables is preferred and therefore recommended in the context. This is because, according to the well-known principle of Occam's razor, whereby "all factors being equal, the simplest explanation is preferred", it is convenient to choose a mathematical approach that balances the explanatory power and the simplicity of the model. We also point out that the use of the oversampling technique called SMOTE was necessary in the training phase, as the original data set was so heavily unbalanced that the analysis could not have been conducted otherwise.
Potential future work could include the analysis of other features in addition to those considered here, such as time series that track the status of

the customer's bank account, or the creation of new features that reflect the interaction between variables. In addition, an alternative feature selection method could be experimented with, using either another filter method or a wrapper or embedded method, and comparing them. A dimensionality reduction technique could also be implemented, using a Principal Component Analysis (PCA), because first of all it would be interesting to study the variability of the features themselves and then select the main components on the basis of these considerations.

It would also be interesting to make a study concerning what metrics are the most relevant for this type of problem. As mentioned previously, in this project the main metric all evaluations were analyzed by was F-score, because the aim was to achieve a trade-off between the sensitivity and the precision. If a deeper analysis could be performed regarding the most relevant metric for this type of problem, then potentially a weight function could be implemented if one of the metrics explored turned out to be of more importance. The example of a weight function can be to use a weighted F-score, where β is not set to 1, but the value of interest. One could also engage in a cost-sensitive learning approach, which is a subfield of machine learning that takes into account the costs of prediction errors (and potentially other costs) when training a machine learning model. It is a field of study that is closely related to the field of unbalanced learning that deals with classification on datasets with an asymmetric distribution of classes.

It would be worth also pursuing the analysis of the economic impact of choosing one model over another when a bank has to decide whether or not to give a loan to a client, that is dealing with the model risk concept. Thus, one could calculate the misclassification error cost for each model and make a comparative analysis of the results at varying false positive and false negative cost settings, respectively. To this study it would be interesting to add the calculation of the expected maximum profit as a utility function for the bank. Finally, it would be interesting to study the optimisation of the mortgage decision process by the bank, looking for the optimal trade-off between

the misclassification error cost and the maximum profit.

Bibliography

[1] Shai Shalev-Shwartz and Shai Ben-David, *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, 2014.

[2] Ian Goodfellow, Yoshua Bengio, Aaron Courville, *Deep Learning*, https://www.deeplearningbook.org/.

[3] Trevor Hastie, Robert Tibshirani, Jerome Friedman, *The Elements of Statistical Learning*, Second Edition, Springer.

[4] Tianqi Chen, Carlos Guestrin, *XGBoost: A Scalable Tree Boosting System*

[5] Chawla Nitesh v. et al., *SMOTE: Synthetic Minority Over-sampling Technique* in *Journal of Artificial Intelligence Research* 16 (2002)

[6] Jean D. Gibbons, *Nonparametric Measures of Association*, Thousand Oaks, California: SAGE Publications, Inc.

[7] Cyril Goutte and Eric Gaussier, it A probabilistic interpretation of precision, recall and F-score, with implication for evaluation, in *European Conference on Information Retrieval*, Springer 2005.

[8] Camilla Calì and Maria Longobardi, *Some mathematical properties of the ROC curve and their applications*, in *Ricerche di Matematica* 64 (Oct. 2015).

[9] Stuart J. Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach* , Upper Saddle River, New Jersey 07458: Pearson Education, Inc, 2010.

[10] Rok Blagus and Lara Lusa, *Joint use of over- and under-sampling techniques and cross validation for the development and assessment of prediction models*, in *BMC Bioinformatics* 16 (2015).

[11] Chamboko, R., Bravo, J. M. (2016), *On the Modelling of Prognosis from Delinquency to Normal Performance on Retail Consumer Loans*, Risk Management 18 (4)

[12] Chamboko, R., Bravo, J. M. (2018c), *A multi-state approach to modelling intermediate events and multiple mortgage loan outcomes*, Working Paper, submitted to Journal of Banking and Finance.

[13] Durand, D. (1941). *Risk Elements in Consumer Instalment Financing.* (D. Durand, Ed.). National Bureau of Economic Research. Retrieved from http://www.nber.org/books/dura41-1

[14] Thomas, L., Edelman, D. Crook, J. (2002). *Credit Scoring and Its Applications. Mathematical Modeling and Computation.* Society for Industrial and Applied Mathematics.

[15] Banasik, J., Crook, J. N. Thomas, L. C. (1999). *Not if but when will borrowers default.* Journal of the Operational Research Society, 50(12), 1185–1190. https://doi.org/10.1057/palgrave.jors.2600851 https://doi.org/doi:10.1137/1.9780898718317

[16] Hand D.J Jacka S. (1998). *Statistics in Finance.* Hand D.J. and Jacka S. (eds.) Statistics in finance, Edward Arnold

[17] Henley, W.E. (1995) *Statistical Aspects of Credit Scoring.* Ph.D. Thesis, Open University, Milton Keynes.

[18] Basel Committee on Banking Supervision, *An Explanatory Note on the Basel II IRB Risk Weight Functions*, Bank for International Settlements

[19] Gaffney et al, *A transitions-based framework for estimating expected credit losses*,Financial Stability Division Central Bank of Ireland.

[20] G. James, D. Witten, T. Hastie, R. Tibshirani, *An Introduction to Statistical Learning*, Springer 2017.

[21] B. Zhu, W. Yang, H.Wang, Y. Yuan, *A Hybrid Deep Learning Model for Consumer Credit Scoring*, 2018 International Conference on Artificial Intelligence and Big Data

[22] J. Wu, *Introduction to Convolutional Neural Networks*, National Key Lab for Novel Software Technology, Nanjing University, China.

[23] Elena Dumitrescu, Sullivan Hué, Christophe Hurlin, Sessi Tokpavi. *Machine Learning or Econometrics for Credit Scoring: Let's Get the Best of Both Worlds.*, 2020. ffhal-02507499v2f

[24] Hurlin, C., Leymarie, J., and Patin, A. (2018). Loss functions for loss given default model comparison. European Journal of Operational Research

[25] Viaene, S. and Dedene, G. (2004). Cost-sensitive learning and decision making revisited. European Journal of Operational Research.

[26] Akkoc, S. (2012). An empirical comparison of conventional techniques, neural networks and the three stage hybrid adaptive neuro fuzzy inference system (ANFIS) model for credit scoring analysis: The case of Turkish credit card data. European Journal of Operational Research

[27] West, D. (2000). Neural network credit scoring models. Computers Operations Research, 27(11-12)

[28] Lessmann, S., Baesens, B., Seow, H.-V., and Thomas, L. C. (2015). Benchmarking stateof-the-art classification algorithms for credit scoring: An update of research. European Journal of Operational Research

[29] Thomas, L., Crook, J., and Edelman, D. (2017). Credit scoring and its applications. SIAM

[30] Verbraken, T., Bravo, C., Weber, R., and Baesens, B. (2014). Development and application of consumer credit scoring models using profit-based classification measures. European Journal of Operational Research