POLITECNICO DI TORINO

Master Course
in Mathematical Engineering

Master Thesis

Credit Risk Assessment Using Machine Learning Techniques

Supervisor
Prof. Patrizia Semeraro

Candidate
Martina Scagliola

Academic Year 2021-2022

Summary

Credit is a must in financial systems. For all financial institutions, whose role is to allocate credit, it is necessary to fully understand the risk behind it and to correctly decide who to give credit and who not. To do so, they make use of credit scoring, which is one of the most successful application of statistical and operational research modelling in finance.
The aim of this thesis is to combine supervised and unsupervised machine learning models to predict the probability of default of a set of individuals who asked a loan to a bank, and to correctly classify them according to their individual propensity to default.
In Chapter 1 we introduce the concepts of credit risk and credit scoring, and then we formally describe two mixture models: Bernoulli and Poisson mixture model. Chapter 2 illustrates the fundamental concepts behind machine learning and contains the theoretical description of some supervised learning models (logistic regression, support vector machine, K-nearest neighbors, random forest and AdaBoost classifier) and some clustering methods. In Chapter 3 and Chapter 4 we perform a credit score analysis on a public data set which simulate the real data set of a bank. In particular, we test the validity of the integration of unsupervised and supervised machine learning techniques by comparing individual models and cluster-based models performances. Chapter 3 contains the preprocessing part of the analysis, while the following Chapter focuses on the application of the machine learning models and the evaluation of their performances according to a set of measures such as AUC, accuracy, F-score and type I and type II errors.

In particular, it introduces the concept of expected misclassification cost, which give us an idea of the economic impact of models results.

Contents

List of Tables 8

List of Figures 9

1 Credit Risk 11

1.1 Credit Risk Assessment . . . 13
1.2 Credit Scoring History . . . 14
1.3 Credit Risk Modeling . . . 16
1.4 Mixture Models . . . 17
1.4.1 Bernoulli mixture model . . . 17
1.4.2 Poisson mixture model . . . 18

2 Machine Learning 19

2.1 PAC Learning Model . . . 20
2.2 n-Fold-Cross-Validation . . . 22
2.3 Assessing Model Accuracy . . . 24
2.3.1 Bias-variance trade-off . . . 25
2.3.2 Bias-variance decomposition for L_C . . . 27
2.4 Supervised Learning for Binary Classification . . . 28
2.5 Logistic Regression . . . 29
2.6 Support Vector Machine . . . 31
2.6.1 Hard-SVM . . . 31
2.6.2 Support vectors . . . 34

2.6.3 Soft-SVM . . . 34
2.6.4 Kernels . . . 35
2.7 K-Nearest Neighbors . . . 38
2.8 Decision Trees . . . 40
2.8.1 Recursive binary splitting . . . 42
2.8.2 Grow-then-prune strategy . . . 43
2.9 Weak Learning and Ensemble Methods . . . 43
2.9.1 Bagging . . . 44
2.9.2 Random forest . . . 45
2.9.3 Boosting . . . 45
2.9.4 AdaBoost . . . 46
2.10 Performance Measures . . . 48
2.11 Unsupervised Learning . . . 51
2.12 Clustering . . . 51
2.12.1 Cost minimization clusterings . . . 53
2.12.2 Elbow and Silhouette methods . . . 54

3 Application on data 57
3.1 Motivation . . . 57
3.2 Data Set Description . . . 57
3.3 Data Exploration . . . 59
3.4 Data Preprocessing . . . 64
3.4.1 Missing values . . . 64
3.4.2 Outliers detection . . . 65
3.4.3 Dataset encoding . . . 66
3.4.4 Oversampling with SMOTE . . . 66
3.4.5 Features selection . . . 68

4 Models Settings and Performances 71
4.1 Individual models . . . 71
4.2 Cluster-based models . . . 75
4.3 ROC Curve Analysis . . . 76

4.4 Expected Misclassification Cost 77

5 Conclusion 83

Bibliography 85

List of Tables

2.1 Confusion matrix. 48
3.1 Dataset description 59
3.2 Percentage of missing values for each feature. 65
4.1 Individual models performances on entire data set. 74
4.2 Individual models performances on features selected data set. 74
4.3 Cluster-based models performances based on features selected data set. 76
4.4 A 2D misclassification cost matrix (*positive* bad creditors, *negative* good creditors). 78

List of Figures

2.1 Left: training error and test error as functions of model complexity. Right: bias-variance decomposition of the test error. In both figures the vertical line indicates the complexity level corresponding to the smallest test error. 25

2.2 On the left: a too simple model trained on two distinct data sets S_1 and S_2. This model suffer of high bias. On the right: a too complex model trained on the same two data sets. This model suffer of high variance. 26

2.3 Maximum-margin hyperplane and margins for an SVM trained with samples from two classes. 32

2.4 SVM with different kernels. The decision boundary is non-linear but the underlying problem is for a linear separating hyperplane. 39

2.5 Left: scheme of a decision tree with numerical questions based on X_1 and X_2. Right: Partition of the two-dimensional space induced by that tree. 41

2.6 ROC curve example for four different models. Model D is completely uninformative, while model A represent the idel one. 51

3.1 Count of the two classes default=0 and default=1. 60

3.2 Proportion of default and non default individuals for different classes of categorical variables. 61

3.3 Box plots for quantitative variables. 63

3.4 Correlation heatmap. 64
3.5 Data set outliers (in red). 66
3.6 Correlation heatmap for encoded data. 67
3.7 Example of SMOTE procedure. 68
4.1 Comparison between performance measures for individual models (in blue) and for cluster-based models (in orange) on feature selected data set. 77
4.2 From top to bottom: ROC curves for individual models based on entire data set, ROC curves for individual models based on features selected data set and ROC curves for cluster-based models on features selected data set. 80
4.3 Expected misclassification error both for individual models (in blue) and for cluster-based models (in orange), applied on feature selected data set. 81

Chapter 1

Credit Risk

Credit is an absolute must in the financial system, affects everyone and drives the global economy. Credit allows individuals to finance their needs of acquiring a house, car, furniture etc., assists companies to start or expand their business, and enables governments to finance public interest projects. If managed well, it can build an economy, produce an efficient allocation of capital and wealth and bring prosperity.

The allocation of credit is performed by financial intermediaries such as commercial and investment banks, saving and loan associations, insurance companies, mutual funds, pension funds and finance companies. They are crucial to the healthy functioning of financial markets because of their role in deciding who gets credit and at which price.

Over the past three decades intermediaries begin to offer increasingly sophisticated products and innovative financial contracts. However, if their risk is not fully understood, they can lead to devastating repercussion on the financial system.

After the crisis of a German bank (Herstatt Bank), in 1974 the central bank governors of the Group of Ten countries estabilished the Basel Committee on Banking Supervision (BCBS). Since the issuance of the first Concordat of 1975, the Committee, constituted by representatives of central banks and banking supervisory bodies, dictated international standards which aim is

to ensure banking regulation. The agreements have occurred over time: Basel I in 1988 and Basel II in 2004. Then, after the financial crisis of 2007, the importance of credit and credit management has increased and even if financial intermediaries have always been regulated, their regulation has had to change dramatically. In fact, based on the traumatic experience of the crisis, in 2010 the Committee published the first text of Basel III. This accord introduced stronger risk management requirements for banks.
To better understand what *credit risk* is we can give a definition. Credit risk is the risk of financial loss due to the borrower's bond issuer's or counterparty's (the obligors) failure to honour their financial obligations. This can be due to the inability or unwillingness of the counterparty, but this last case is less common with respect to the first one. This inability is linked with the concept of *default*. Default can be defined as a missed or delayed payment of a contractual obligation or legal receivership of the obligor that will probably cause one or more missed or delayed future payment(s).
The sources of credit risk can be various (deposits, prepayment of goods or services, contingent claims, bonds, derivatives etc.), but the focus of our analysis will be on loans.
A *loan* is a cash outflow provided from the lender to the borrower with the promise to repay it back at a later scheduled date. Of course, the loan has a cost which is defined as the interest rate paied by the borrower to the lender on the loan principal amount at scheduled interest payment dates. The full term and conditions of the loan are defines in the loan agreement. Going into more detail, loans can be of different types: secured or unsecured. As the name says, a secure loan imply a lower credit risk for the lender than an unsecured one. This is due to the fact that with a secured loan the borrower pledges asset as collateral that can be, in case of necessity, repossessed and sold by the lender to recover the sums owed. Classical examples of secured personal loans are mortgages or car loan, which are respectively secure on the house or on the car. On the other hand unsecured loans do not involve any type of collateral. Common examples include credit cards,

personal loans etc..

1.1 Credit Risk Assessment

It is quite clear at this point that commercial banks and financial institutions have to deal with a fundamental decision: whether or not to grant a loan to a customer. To answer this question they make use of *credit scoring*, which is one of the most successful applications of statistical and operations research modeling in finance.
Credit scoring is the set of decision models and their underlying techniques that aid lenders in the granting of consumer credit. These techniques decide who will get credit, how much credit they should get, and what operational strategies will enhance the profitability of the borrowers to the lenders. They access, in some way, the risk in lending to a particular consumer.
It is important to note that creditworthiness is not an attribute of individuals as weight, age or even income. It is an assessment by a lender of a borrower and reflects the circumstances of both the party's view of the likely future economic scenarios. Thus, the same individual can be classified as creditworthy from some lenders but not from others. However, a possible danger of credit scoring is that this may cease to be the case and there will be those who can get credit from all lenders and those who can not get credit at all.
Each customer i is assigned to certain risk class $Y_i \in \{0,1\}$ based on his individual propensity to default on payments. In particular, $Y_i = 1$ indicate the default class while $Y_i = 0$ indicate the non-default class. The likelihood of a default over a specific time horizon $[t, t+\delta t]$ is called *default probability* (DP) and is indicated with $p_i = P(Y_i = 1)$. The default indicator Y_i is thus a random variable with distribution

$$Y_i \sim Bernoulli(p_i). \tag{1.1}$$

The probabilities p_i can be obtained by data on previous customers, with their application details and subsequent credit history available. All the

techniques use the sample to identify possible connections between the charateristics of the costumer and how 'good' or 'bad' their subsequent history is. Note that even if terms as 'uncreditworthy' or 'bad' can be interpreted with a negative connotation, in reality they only want to state that lending to this customer represent a risk that the lender is not willing to take.
Let us now suppose to have a portfolio of m loans. Its credit loss is given by the sum of individual loss

$$L_m := \sum_{i=1}^{m} e_i \cdot l_i \cdot Y_i, \tag{1.2}$$

where

- $l_i \in (0,1]$ represent the loss rate of i and $\sum_{i=1}^{m} l_i = 1$
- $e_i > 0$ represent the loan amount and, without loss of generality, can be normalized to 1.

If we assume $l_i = 1/m$ we get $L_m = \frac{S_m}{m}$ where

$$S_m = \sum_{i=1}^{m} Y_i \tag{1.3}$$

is the number of insolvencies.
In practice, assuming the same loss rate for every i, in order to analyze the total loss L_m it is sufficient to study the behaviour of S_m.

1.2 Credit Scoring History

The history of credit scoring is only 70 years old. The first idea of solving the problem of identifying groups in a population was introduced in statistics by Fisher (1936). Even if the context was quite different, his idea was very similar to the classification idea at the base of credit scoring. The first one to recognize that the same approach could be used to discriminate between good and bad loans was Durand in 1941.

During the 1930s, some companies has introduced numerical scoring analysis to overcome the inconsistencies in credit decision across credit analysts. Then, with the start of the World War II, all the finance houses began to experience difficulties with credit management. The problem was that many credit analysts were being drafted into military service, hence the firms had the analysts write down the rules of thumb they used to decide to whom to give loans; these rules where then used by nonexperts to help make credit decisions. They were both numerical scoring systems already introduced and sets of conditions that needed to be satisfied.
After the war ended, it did not take long for some folks to connect the automation of credit decision and the classification techniques being developed in statistics and to see the benefit of using statistically derived models in lending decisions (Wonderlic 1952). In the late 1960, with the arrival of credit cards, the banks and many other credit card issuers realize the usefulness of credit scoring. In fact the number of people applying for credit cards each day made it impossible to do anything but automate the lending decisions. Also thanks to the growth in computing power made all this possible, the default rates dropped by 50% or more.
The only opposition came from those like Capon (1982), Associate Professor of Business at Columbia University, who believed that "the brute force empiricism of credit scoring offends against the traditions of our society".
The complete acceptance of credit scoring was due to the passage of the Equal Credit Opportunity Acts and its amendments in the U.S. in 1975 and 1976. In 1975 the Congress prohibited discrimination in the granting of credit on the basis of sex and marital status. The following year the Act was amended to include race, color, religion, national origin and receipt of income from a public assistance program as proscribed charateristics.
Given the success of credit scoring in credit cards, in the 1980s, the banks started using scoring for other products like personal loans. In the 1980s, logistic regression and linear programming were introduced. More recently,

the development of artificial intelligence has led to usage of more sophisticated machine learning models that can replace the logistic regression one. Some of these methods, such as SVM, KNN, Random Forests etc., will be better introduced in Chapter 2. However, despite the advances and applications of machine learning models in credit scoring, it is important to take into consideration that some of these methods suffer from a major issue: while linear regression is known for its simplicity and transparency in predictions, some machine learning models are incapable to explain predictions.

1.3 Credit Risk Modeling

The development of the market for credit derivatives and the Basel III process has generated a lot of interest in quantitative credit risk models, so credit risk modelling is a very active subfield of quantitative finance and risk management.
In this section we provide a brief overview of the various model types that are used in credit risk, focusing on static models. In fact, credit risk management models are used to determine the loss distribution of a loan (or a bond) portfolio over a fixed time period (typically at least one year), and to compute loss-distribution-based risk measures. Hence these models are typically static, in the sense that they focus on the loss distribution for the fixed time period rather than a stochastic process describing the evolution of risk in time.
Credit risk models can be divided into *structural* or *firm-value models* on the one hand and *reduced-form models* on the other. In firm-value models default occurs whenever a stochastic variable, representing an asset value, falls below a threshold representing liabilities. For these reasons static structural models are often called *threshold models*.
In reduced-form models, the mechanism leading to default is left unspecified. The default time is modelled as a non-negative random variable, whose

distribution typically depends on economic covariables. In next section we will focus on *mixture models*, which can be thought of as static portfolio version of reduced-form models. More precisely, mixture models assume conditional independence of defaults given common underlying stochastic factors.

1.4 Mixture Models

In a mixture model the default risk of an obligor is assumed to depend on a set of common economic factors, such as macroeconomic variables, which are also modelled stochastically. As anticipated before, given a realization of the factors, defaults of individuals are assumed to be independent. Dependence between defaults stems from the dependence of individual default probabilities on the set of common factors.

1.4.1 Bernoulli mixture model

Definition 1. *Given some $n < m$ and a n-dimensional random vector $\boldsymbol{X} = (X_1, ..., X_n)'$, the random vector $\boldsymbol{Y} = (Y_1, ..., Y_m)'$ follows a Bernoulli mixture model with factor vector $\boldsymbol{X}$ if there are functions $\hat{p}_i : \mathbb{R}^n \to [0,1], 1 \leq i \leq m$, such that conditional on $\boldsymbol{X}$ the components of $\boldsymbol{Y}$ are independent Bernoulli random variables satisfying*

$$P(Y_i|\boldsymbol{X} = \boldsymbol{x}) = \hat{p}_i(\boldsymbol{x}).$$

Given a vector $\mathbf{y} = (y_1, ..., y_m)'$ in $0,1^m$ we have that

$$P(\mathbf{Y} = \mathbf{y}|\mathbf{X} = \mathbf{x}) = \prod_{i=1}^{m} \hat{p}_i(\mathbf{x})^{y_i}(1 - \hat{p}_i(\mathbf{x}))^{1-y_i}. \tag{1.4}$$

If we integrate 1.4 over the distribution of the factor vector $\mathbf{X}$ we get the unconditional distribution of the default indicator vector $\mathbf{Y}$. In particular, the default probability of individual i is given by $p_i = P(Y_i = 1) = E(\hat{p}_i(\mathbf{X}))$. In fact,

$$p_i = P(Y_i = 1) = E(Y_i) = E(E(Y_i|\mathbf{X}) = E(1 \cdot P(Y_i = 1|\mathbf{X}) + 0 \cdot P(Y_i = 0|\mathbf{X}))$$

$= E(P(Y_i = 1|\mathbf{X})) = E(\hat{p}_i(\mathbf{X}))$.

1.4.2 Poisson mixture model

Since default is a rare event, we can think of substituting Bernoulli random variables with Poisson random variables.
In this case an individual may potentially default more than once in the period of interest, even if with a very low probability. We introduce the random variable $\tilde{Y}_i \in \{0,1,2,...\}$ which counts the number of defaults of i. Similarly to before, we can give the following definition.

Definition 2. *Given p and $\boldsymbol{X}$ defined as in section 1.4.1, the random vector $\tilde{\boldsymbol{Y}} = (\tilde{Y}_1, ..., \tilde{Y}_m)'$ follows a Poisson mixture model with factors $\boldsymbol{X}$ if there are functions $\hat{\lambda}_i : \mathbb{R}^n \to (0, \infty), 1 \leq i \leq m$, such that conditional on $\boldsymbol{X} = \boldsymbol{x}$ the random vector $\tilde{\boldsymbol{Y}}$ is a vector of independent Poisson distributed random variables with rate parameter $\lambda_i(\boldsymbol{x})$.*

For small Poisson parameters λ_i, we can approximate the value of the number of default companies S_m with the random variable

$$\tilde{S}_m = \sum_{i=1}^{m} \tilde{Y}_i.$$

Given the factors, $\tilde{S}_m$ is the sum of independent Poisson variables and therefore its distribution satisfies

$$P(\tilde{S}_m = s|\mathbf{X} = \mathbf{x}) = exp\left(-\sum_{i=1}^{m} \lambda_i(\mathbf{x}) \right) \frac{(\sum_{i=1}^{m} \lambda_i(\mathbf{x}))^s}{s!}. \tag{1.5}$$

Bernoulli and Poisson mixture models are strongly related, in fact if $\tilde{\mathbf{Y}}$follows a Poisson mixture then $\mathbf{Y}$ such that $Y_i = 1_{\{\tilde{Y}_i \geq 1\}}$ for all $i = 1, .., m$, follows a Bernoulli mixture model. In particular, the mixing variables are related by

$$\hat{p}_i(\cdot) = 1 - exp(-\lambda_i(\cdot)).$$

Chapter 2

Machine Learning

Machine learning (ML) is a branch of artificial intelligence that can be defined as the study of efficient algorithms that automatically extract valuable information from data. Another way to see machine learning is as a set of computational methods which use experience to improve performance or to make accurate predictions. Here, the word 'experience' refers to past information available to the learner: this means data. So the crux of machine learning is data. In particular data quality and data size are at the core of machine learning and, since the success of a learning algorithm depends on the data used, machine learning is strictly related to data analysis and statistics.
Learning is a very wild domain, consequently it can be ranched into subfields dealing with different types of learning. The most common partition is the one that distinguish between supervised and unsupervised learning according to the types of training data available to the learner.

- In **Supervised learning** the learner has a set of labeled examples and want to make predictions for all unseen points. This is the most common scenario when dealing with classification or regression problems.

- In **Unsupervised learning** the learner can only count on a set of

unlabeled examples to makes predictions for all unseen points. Clusters and dimensionality reduction problems are examples of this type of learning problems.

Both supervised and unsupervised machine learning have been deeply extensively applied in credit risk assessment. In this work we will first focus on supervised learning techniques for *classification* task. In our case, as anticipated in Section 1.1, the classes are two, so we will develop algorithm for the so-called *binary classification*.
Secondly, we will introduce some unsupervised learning techniques, such as *clustering*, that can be used as complementary tools to supervised ones.

2.1 PAC Learning Model

We will now introduce some terminology and notation that will be used in the next chapters.

- We denote by X the set of all possible *instances* of data used for learning (also called *examples*). X is also sometimes referred to as the *input space*.
- The *features* are the attributes associated to an example.
- The *labels* (or *target values*) are the categories assigned to examples. The set of all possible labels is denoted by Y. In the context of binary classification we can assume Y= {0,1}.
- An *hypothesis set* H is a set of functions $h : X \to Y$ mapping feature to the set Y of labels.
- A *concept* $c : X \to Y$ is a mapping from X to Y. Since $Y = \{0,1\}$, we can identify c with the subset of X over which it takes the value 1.
- A *concept class* C is a set of concepts we may wish to learn.

Let us assume to have a set of independently and identically distributed (i.i.d.) examples, according to a fixed but unknown distribution D. The learner consider a set of possible concepts H, which may not coincide with C. He receives a sample $S = (x_1, ..., x_m)$ and the labels $(c(x_1), ..., c(x_m))$, which are based on the target concept $C \in C$ to learn. The objective is, of course, to find an hypotheses $h \in H$ which has a small generalization error with respect to c, using the labeled sample S. The *generalization error* of an hypotheses $h \in H$ is

$$R(h) = \underset{x \sim D}{P}[h(x) \neq c(x)] = \underset{x \sim D}{E}[1_{h(x) \neq c(x)}]. \tag{2.1}$$

This error is not directly accessible to the learner, since both D and c are unknown. The learner can only measure the *empirical error*

$$\hat{R}(h) = \frac{1}{m} \sum_{i=1}^{m} 1_{h(x_i) \neq c(x_i)} \tag{2.2}$$

which is the average error of h over the sample S. Note that, for a fixed $h \in H$, the expectation of the empirical error based on an i.i.d. sample S is equal to the generalization error: $E[\hat{R}(h)] = R(h)$. This follows directly from the linearity of the expectation and from the fact that the ample is drawn i.i.d., in fact

$$\underset{S \sim D^m}{E}[\hat{R}(h)] = \frac{1}{m} \sum_{i=1}^{m} \underset{S \sim D^m}{E}[1_{h(x_i) \neq c(x_i)}] = \frac{1}{m} \sum_{i=1}^{m} \underset{S \sim D^m}{E}[1_{h(x) \neq c(x)}]$$

for any x in S. Thus,

$$\underset{S \sim D^m}{E}[\hat{R}(h)] = \underset{S \sim D^m}{E}[1_{h(x) \neq c(x)}] = \underset{x \sim D}{E}[1_{h(x) \neq c(x)}] = R(h).$$

We can now introduce the *Probably Approximately Correct* (PAC) learning concept.

Let $size(c)$ be the maximal cost of the computational representation of $c \in C$.

Definition 3. *A concept class C is said to be PAC-learnable if there exists an algorithm A and a polynomial function poly$(\cdot, \cdot, \cdot, \cdot)$ such that for any*

$\epsilon > 0$ and $\delta > 0$, for all distributions D on X and for any target concept $c \in C$, the following holds for any sample size $m \geq poly(1/\epsilon, 1/\delta, n, size(c))$:

$$\underset{S \sim D^m}{P}[R(h_S) \leq \epsilon] \geq 1 - \delta.$$

If A further runs in $poly(1/\epsilon, 1/\delta, n, size(c))$, then C is said to be efficiently PAC-learnable. When such an algorithm A exists, it is called a PAC-learning algorithm for C.

In other words, a concept class C is said to be PAC-learnable if the hypothesis returned by A after observing a number of points polynomial in $1/\epsilon$ and $1/\delta$ is approximately correct (the error is at most ϵ) with high probability (i.e. with probability at least $1 - \delta$).

2.2 n-Fold-Cross-Validation

Before applying any machine learning algorithm, data should be split in distinct sets: training set, validation set and testing set.

- The *training set* (or *learning set*) is the set of examples used to train a learning algorithm.
- The *validation set* is the set of examples used to tune the parameters $\boldsymbol{\theta}$ of a learning algorithm when working labeled data (i.e. for *model selection*).
- The *test set* is the set of examples used to evaluate the performances of a learning algorithm. This set is separate from the training and validation data and is not made available in the learning stage.

The problem is that, in practice, the amount of labeled data available is often too small to set aside a validation sample. In fact, that would leave an insufficient amount of training data. To overcame this problem, a widely adopted method is *n-fold-cross-validation*, which is used to exploit the labeled data both for selection of the free parameters of the algorithm and

for training.

Given the training sample $S = ((x_1, y_1), (x_2, y_2), ..., (x_m, y_m))$ of m labeled independent examples, the method consists of first fixing a value of $\boldsymbol{\theta}$, and then randomly partitioning S into n subsamples of approximately equal size. The ith subsample (or *fold*) is thus a labeled sample $((x_{i1}, y_{i1}), ..., (x_{im_i}, y_{im_i}))$ of size m_i.

Then, for any $i \in [1, n]$, the algorithm is trained on all but the ith fold in order to generate a hypothesis h_i. Then the performance of h_i is tested on the ith fold. The goodness of the paramenter $\boldsymbol{\theta}$ is evaluated according to the so-called *cross-validation error* $\hat{R}_{CV}(\boldsymbol{\theta})$, defined as follows

$$\hat{R}_{CV}(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{m_i} \sum_{j=1}^{m_i} L(h_i(x_{ij}), y_{ij}) \tag{2.3}$$

where L is the *loss function* that measures the difference between a predicted label and a true one.

The most common loss function used in the context of classification is the *misclassification loss* defined as

$$L_C(y, h(x)) = 1_{h(x) \neq y}. \tag{2.4}$$

So, in practice, the quantity $\hat{R}_{CV}(\boldsymbol{\theta})$ is the average error of the n hypothesis h_i.

A common choice for m_i is $m_i = m/n \quad \forall i \in [1, n]$. The problem now is the choice of n. The appropriate choice is subject to the bias-variance tradeoff, a concept that will better explained in Section 2.3.1. For a large n, each training sample has size close to the size m of the full sample and the training sample are quite similar. In contrast, for small values of n each training sample has size significantly less than m and the training samples are more diverse. In the first case the method tends to have a small bias but a large variance. On the contrary, in the second one, the method tends to have a smaller variance but a larger bias.

In machine learning applications, n is typically chosen equal to 5 or 10, but this is not always the case. A typical example is *leave-one-out cross-validation*, in which $n = m$. In general, the leave-one-out error is very

costly to compute, because it requires training n times on sample of size $m-1$.
Cross-validation is commonly used both for model selection and performance evaluation.
In case of model selection, n-fold-cross-validation is applied as follows:

- the full data set is split into a training and a test sample
- the training sample is used to compute $\hat{R}_{CV}(\boldsymbol{\theta})$ for a certain number of possible value of $\boldsymbol{\theta}$ ($\boldsymbol{\theta} \in \Theta$)
- $\boldsymbol{\theta}$ is set to the value $\boldsymbol{\theta}_0$ such that

$$\boldsymbol{\theta}_0 = arg \min_{\boldsymbol{\theta} \in \Theta} \hat{R}_{CV}(\boldsymbol{\theta}) \tag{2.5}$$

- the algorithm is trained over the full training sample of size m with parameter setting $\boldsymbol{\theta}_0$.

We will better see in the next section how cross-validation can be used for performance evaluation.

2.3 Assessing Model Accuracy

In order to evaluate the performance of a learning method on a given data set, we need some way to measure how well predictions match the observed data. In other words we need to quantify the extent to which the predicted response value for a given observation is close to the true response value for that observation.
A general problem of supervised learning is often formulated as an optimization problem in which we wish to minimize the error of hypotheses h over the training set. However, in general, we are interested in a model that generalizes well on unseen data rather than a model that perfectly fits the training samples. Unfortunately, there is a fundamental problem: there is no guarantee that good results on the training set imply good performances on the testing set.

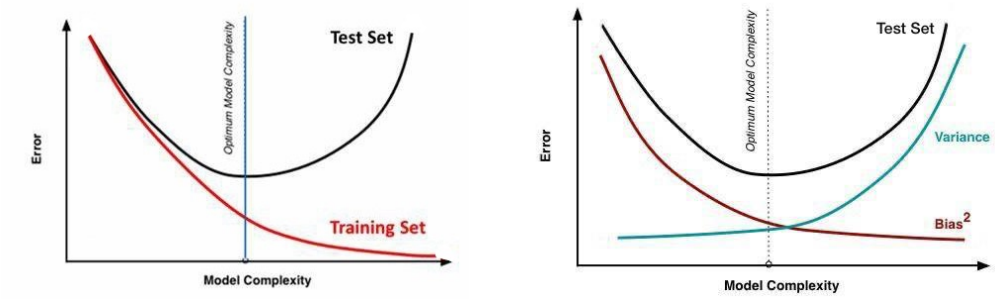

Figure 2.1. Left: training error and test error as functions of model complexity. Right: bias-variance decomposition of the test error. In both figures the vertical line indicates the complexity level corresponding to the smallest test error.

The left-hand panel of Figure 2.1 illustrates this phenomenon. The red curve represents the training error as a function of the model complexity and declines monotonically as complexity increases. The test error is displayed by the black curve. The test error initially declines as the level of flexibility increases but, at some point, it levels off and then starts to increase again. This is a fundamental property of statistical learning that holds regardless of the particular data set and regardless of the method being used. When a given method yields a small training error but a large test error, we are said to be *overfitting* the data.
The optimal model is therefore the one with minimal test error. If no training set is available, cross-validation can be used to estimate the test error using only training data.

2.3.1 Bias-variance trade-off

The U-shape of test error curve is due to the fact that test error can be decomposed into the sum of three fundamental quantities: the *residual error* σ (or *minimal attainable error*), the *systematic error* (or *bias*) and the effect of the *variance*. The exact decomposition depends on the loss function L.
We will now better explain these three quantities and then we will find the

exact decomposition for the misclassification loss L_C.

- The *residual error* is the error obtained by the best possible model. It provides a theoretical lower bound that is independent of the learning algorithm. It only depends on the problem and on the loss criterion used.

- *Bias* refers to the error that is introduced by approximating real-life problem, which may be extremely complicated, by a much simpler model.

- *Variance* refers to the amount by which the model h would change if we estimated it using different training sets. Ideally the model should not vary too much between training sets.

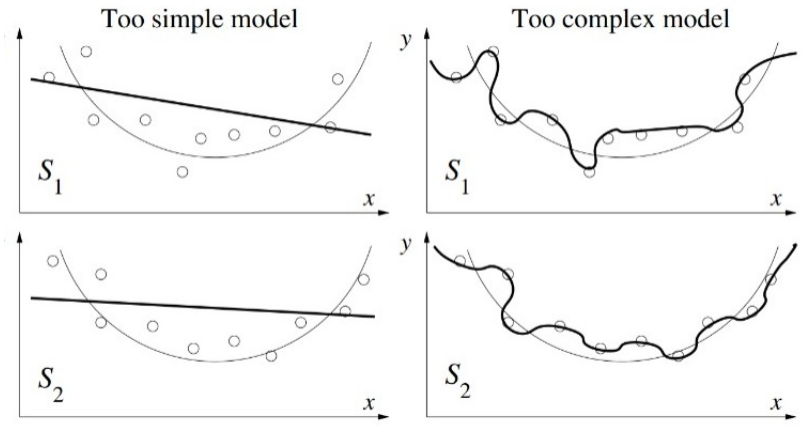

Figure 2.2. On the left: a too simple model trained on two distinct data sets S_1 and S_2. This model suffer of high bias. On the right: a too complex model trained on the same two data sets. This model suffer of high variance.

As a general rule, increasing the flexibility of the method, the variance will increase and the bias will decrease.

2.3.2 Bias-variance decomposition for L_C

Bias-variance decomposition for misclassification loss is not unique, several authors proposed their own one. In this section we will see the most natural one, supported by Domingos(2000). A more complete overview is available in [11].

Let us suppose to have a sample $S = ((x_1, y_1), ..., (x_m, y_m))$ of m randomly independent pairs, drawn from a probability distribution D over $X \times Y$. As said before, the empirical error $\hat{R}(h)$ of an hypothesis $h(x)$ on a sample S can be computed as the expectation

$$\hat{R}(h) = E_{x,y}[L(y, h(x)] \tag{2.6}$$

where L is a generic loss function. The quantity $\hat{R}(h)$ is itself a random variable, hence we are interested in studying its expected value over the sample set S.

$$E_S[\hat{R}(h(x))] = E_S[E_{y|x}[L(y, h(x))]] = P_S(y \neq h(x)|x). \tag{2.7}$$

Let us now on the case in which the loss function is the misclassification loss L_C. In this case the best possible model h^*, called *Bayes model*, is the classifier that assign each observation to the most likely class, given its predictor values

$$h^*(x) = arg \max_{y \in Y} P(Y = y|X = x). \tag{2.8}$$

The corresponding residual error, also called *Bayes error rate*, is

$$R(h^*(x)) = 1 - P(Y = h^*(x)|X = x). \tag{2.9}$$

A direct consequence is that the residual error is given by

$$\sigma_c(x) = 1 - P(Y = h^*(x)|X = x). \tag{2.10}$$

Symmetrically to the Bayes model, we can define the *majority vote classifier*

$$h_{maj}(x) = arg \max_{y \in Y} P_S(h(x) = y) \tag{2.11}$$

which outputs at each point the class receiving the majority of votes among the distribution of classifiers induced from the distribution D of learning sets.
The squared bias is the error of the average model with respect to the best possible model

$$bias_c(x) = L_C(h^*(x), h_{maj}(x)). \tag{2.12}$$

So, biased points are those for which the majority vote classifier disagrees with the Bayes classifier.
Instead, we can define variance as the average error of model h induced from the random sample S with respect to the majority vote classifier:

$$var_c(x) = E_S[L_C(h(x), h_{maj}(x))] = P_S(h(x) \neq h_{maj}(x)). \tag{2.13}$$

Unfortunately, these natural bias end variance terms do not sum up with the residual error to give the misclassification error

$$E_S[\hat{R}(h(x))] \neq \sigma_C(x) + bias_C(x) + var_C(x). \tag{2.14}$$

A quite unintuitive consequence is that increasing the variance may decrease the average classification error in some situations. This is due to the fact that with misclassification loss, much variance can be beneficial because i can lead the system closer to the Bayes classification.

2.4 Supervised Learning for Binary Classification

The supervised machine learning algorithms are used in credit scoring models to find the relationship between the customer features and credit default risk and then predict the default classification usually in a binary format.
We will now describe more in detail, with reference to [8] and [4], some of the most important classification algorithm, such as *logistic regression*, *support vector machine (SVM)*, k-nearest neighbors (*KNN*), *decision trees*,

random forests and *adaptive boosting (AdaBoost).*
Let $S = ((x_1, y_1), ..., (x_m, y_m)) \in (X \times Y)^m$ be a labeled sample of i.i.d. observations drawn according to the distribution D over $X \times Y$, with $x_i \in \mathbb{R}^n$ and $y_i \in Y$ for all $i \in [1, m]$. We will assume S to be the input of each algorithm.
Note that in our context, that is binary classification, Y is a set containing only two classes. For simplicity, only for the mathematical description of SVM and AdaBoost, we will assume $Y = \{-1, +1\}$, instead of $Y = \{0,1\}$ as before.

2.5 Logistic Regression

Rather than modeling Y directly, logistic regression models the probability that Y belongs to a certain category.
From now on we define $p_i := P(Y_i = 1|X_i = x_i)$. We can model these conditional probabilities using linear regression model

$$p_i = \beta^T x_i + \epsilon_i. \tag{2.15}$$

where $\beta \in \mathbb{R}^n$ is the coefficients vector and ϵ_i represent the error.
The problem is that these probabilities must fall between 0 and 1. To avoid this problem we must model p_i using a function that gives outputs between 0 and 1 for all values of x_i.
Logistic regression use the *logistic function*

$$f(x) = \frac{e^x}{1 + e^x} \tag{2.16}$$

to model p_i.
In this way we obtain

$$p_i = \frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}}. \tag{2.17}$$

After a bit of manipulation we get

$$\frac{p_i}{1 - p_i} = e^{\beta^T x_i}. \tag{2.18}$$

The quantity $p_i/(1-p_i)$ is called *odds* and can take value between 0 and ∞. Values of the odds close to 0 and ∞ indicate very low and very high probabilities o default, respectively.
By taking the logarithm of both sides of 2.18 we obtain

$$log\left(\frac{p_i}{1-p_i}\right) = \beta^T x_i. \tag{2.19}$$

The quantity $log(p_i/(1-p_i))$ is called *log-odds* or *logit*.
As we can see from 2.19, the logistic regression model has a logit that is linear in x. This make the model easily interpretable: increasing x_{ij} (the j-th entry of x_i) by one unit changes the log-odds by β_j, or equivalently it multiplies the odds by e^{β_j}. However the relationship between p_i and x_i is not a straight line, so β_j does not represent the change in p_i associated with a one-unit increase in x_{ij}. A one-unit change in x_{ij} will cause a change in p_i which amount depends on the current value of x_{ij}. But regardless of the value of x_{ij}, if β_j is positive then increasing x_{ij} will be associated with increasing p_i and vice versa, if β_j is negative then increasing x_{ij} will be associated with decreasing p_i.

The problem now is how to estimate the coefficients β. The most common method to fit a logistic regression model is *maximum likelihood*. We know that variables Y_i are Bernoulli with parameter p_i and density function

$$P(Y_i = y_i) = p_i^{y_i}(1-p_i)^{1-y_i}. \tag{2.20}$$

Under the assumption that the observations are independent, we can define the *likelihood function* as

$$l(\beta) = \prod_{i=1}^{n} P(Y_i = y_i). \tag{2.21}$$

By substituting 2.19 in 2.20 we get

$$l(\beta) = \prod_{i=1}^{n} \left(\frac{e^{\beta^T x_i}}{1+e^{\beta^T x_i}}\right)^{y_i} \left(1 - \frac{e^{\beta^T x_i}}{1+e^{\beta^T x_i}}\right)^{1-y_i}. \tag{2.22}$$

For simplicity we can take the logarithm on both sides and we get the so called *log-likelihood function*

$$\mathcal{L}(\beta) = log\,(l(\beta)) = \sum_{i=1}^{n} y_i\ log\left(\frac{e^{\beta^T x_i}}{1+e^{\beta^T x_i}}\right) + \sum_{i=1}^{n}(1-y_i)log\left(1-\frac{e^{\beta^T x_i}}{1+e^{\beta^T x_i}}\right)$$

and, after a bit of manipulation, we obtain

$$\mathcal{L}(\beta) = \sum_{i=1}^{n} y_i \beta^T x_i - log(1+e^{\beta^T x_i}). \tag{2.23}$$

The coefficient β can be easily found by solving the convex maximization problem

$$\beta = arg\ \max_{\beta\in\mathbb{R}^n} \mathcal{L}(\beta). \tag{2.24}$$

2.6 Support Vector Machine

The support vector machine (SVM) is a very useful machine learning tool for learning linear predictors in high dimensional futures spaces.
SVM objective is to search for a a halfspace that separates the training set S, according to labels $y_i \in \{\pm 1\}$, with a large margin. This means that each example must be on the correct side of the separating hyperplane but also far away from it.

2.6.1 Hard-SVM

Definition 4. *We say that S is linearly separable if there exists a halfspace (w, b) such that*

$$y_i = sign(< w, x_i > +b), \qquad \forall i = 1, ..., m$$

or equivalently

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

The *margin* of a hyperplane with respect to a training set is defined as the minimal distance between the hyperplane and a point of the training

set. Searching for a hyperplane with a large margin means that, even if we slightly perturb each instance of S, the hyperplane still separate the training set.

The distance between a point x and a hyperplane (w, b) with $||w|| = 1$ is given by

$$d(x, (w, b)) = | < w, x > +b | \tag{2.25}$$

so the closest point in S to the separating hyperplane is

$$\min_{i=1,..,m} | < w, x_i > +b|.$$

Hard-SVM is the learning rule that return a hyperplane that separates the

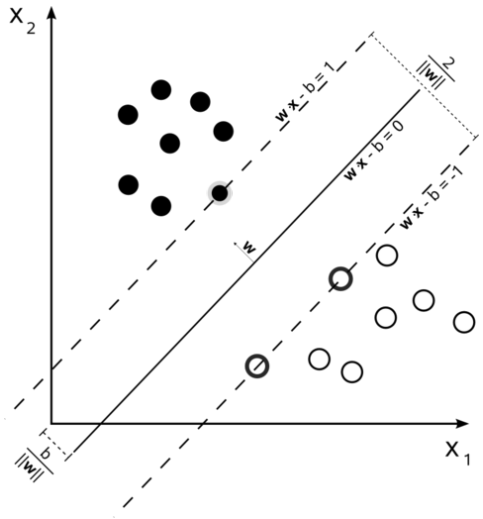

Figure 2.3. Maximum-margin hyperplane and margins for an SVM trained with samples from two classes.

training set S with the largest margin possible. We can formalize this rule as follows

$$\begin{aligned} arg \max_{(w,b):||w||=1} \min_{i=1,...,m} & | < w, x_i > +b | \\ s.t. \quad & y_i(< w, x_i > +b) > 0, \quad \forall i = 1, ..., m. \end{aligned} \tag{2.26}$$

This problem only admits solution under the assumption that S is linearly separable. In this case 2.26 can be reformulated as

$$arg \max_{(w,b):||w||=1} \min_{i=1,...,m} y_i(< w, x_i > +b). \tag{2.27}$$

Furthermore, it is possible to give another equivalent formulation of Hard-SVM rule as a quadratic optimization problem:

- given the training set S, solve

$$\begin{aligned}(w_0, b_0) = &arg \min_{(w,b)} ||w||^2 \\ &s.t. y_i(< w, x_i > +b) \geq 1\end{aligned} \tag{2.28}$$

- output: $\hat{w} = \frac{w_0}{||w_0||}$ and $\hat{b} = \frac{b_0}{||w_0||}$.

This means that finding the largest margin halfspace is equivalent to finding w whose norm is minimal. This result is expressed in the following theorem.

Theorem 1. *The output $(\hat{w}, \hat{b})$ of Hard-SVM rule 2.28 is a solution of problem 2.27.*

Proof. Let (w^*, b^*) be a solution of 2.27 and define the respective margin to be $\gamma^* = \min_{i=1,\dots,m} y_i(< w^*, x_i > +b^*)$. For all i it holds

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

or equivalently

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

Hence, the couple $\left(\frac{w^*}{\gamma^*}, \frac{b^*}{\gamma^*}\right)$ satisfies the constraints of problem 2.28. Therefore, $||w_0|| \leq ||\frac{w^*}{\gamma^*}|| = \frac{1}{\gamma^*}$. As a consequence,

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

for all i. So, since $||\hat{w}|| = 1$, we have that $(\hat{w}, \hat{b})$ is an optimal solution for 2.27.

□

2.6.2 Support vectors

The name 'Support Vector Machine' stems for the fact that the solution w_0 of Hard-SVM is in the linear span of the examples whose distance from the separating hyperplane is exactly $1/||w_0||$. These vectors are called *support vectors*.
This concept is expressed in the so called *Fritz Jhon optimality conditions*:

Theorem 2. *Let w_0 be the solution of Hard-SVM and let*

$$I = \{i : | < w_0, x_i > | = 1\}.$$

Then, there exist coefficients $\alpha_1, ..., \alpha_m$ such that

$$w_0 = \sum_{i \in I} \alpha_i x_i. \tag{2.29}$$

The vectors x_i such that $i \in I$ are the support vectors.

2.6.3 Soft-SVM

Strong-SVM assumes that the training set S is linearly separable, but this is almost never true. A possible solution is to apply Soft-SVM, which is a relaxation of Hard-SVM that can be applied also to non linearly separable training sets. In particular, if S is not linearly separable, constraints in problem 2.28 are never satisfied. Soft-SVM allows the constraints to be violated by introducing nonnegative slack variables $\xi_1, ..., \xi_m$. In this way the new constraints become

$$y_i(< w, x_i > +b) \geq 1 - \xi_i \quad \forall i = 1, ..., m \tag{2.30}$$

Of course, we want the violations of the constraints ξ_i to be as small as possible, so Soft-SVM jointly minimizes the norm of w and the average of ξ_i. This leads to the following optimization problem:

- given the training set S, solve

$$\begin{aligned} \min_{w,b,\xi} \quad & \left(\lambda||w||^2 + \frac{1}{m}\sum_{i=1}^{m}\xi_i\right) \\ s.t. \quad & y_i(< w, x_i > +b) \geq 1 - \xi_i \quad \forall i = 1, ..., m \\ & \xi_i \geq 0 \quad \forall i = 1, ..., m \end{aligned} \tag{2.31}$$

- output: w and b

where $\lambda > 0$ is the parameter that controls the trade-off between the two terms we want to minimize. As before, we can rewrite problem 2.31 in a simpler form. To do this it is necessary to introduce the definition of the *hinge loss*:

$$L_{hinge}((w, b), (x, y)) = max\{0, 1 - y(< w, x > +b)\}. \tag{2.32}$$

Now it is possible to rewrite 2.31 as a regularized loss minimization problem, as follows:

$$\min_{w,b}\left(\lambda||w||^2 + \frac{1}{m}\sum_{i=1}^{m} L_{hinge}((w, b), (x_i, y_i))\right). \tag{2.33}$$

Theorem 3. *Problem 2.31 and 2.33 are equvalent.*

Proof. Let us fix some (w, b) and consider the minimization in 2.31 over ξ. For all i, the best assignment for ξ_i would be

$$\xi_i = \begin{cases} 0 & \text{if } y_i(< w, x_i > +b) \geq 1 \\ 1 - y_i(< w, x_i > +b) & \text{otherwise} \end{cases} \tag{2.34}$$

or equivalently, $\xi_i = L_{hinge}((w, b), (x_i, y_i))$.

□

2.6.4 Kernels

The power of halfspaces is quite restricted, in the sense that most of the training sets are not separable by a halfspaces. To overcome this problem we

can first map the original training set into another space (usually of higher dimension) and then learn an halfspace in that space. More formally, we can proceed as follows:

- We choose a mapping $\phi : X \to \mathcal{F}$, where $\mathcal{F}$ is the *feature spaces*. $\mathcal{F}$ can be any Hilbert space but in general it is taken equal to $\mathbb{R}^n$ for some n.
- We generate from the old training set S a new training set $\hat{S} = ((\phi(x_1), y_1), ..., (\phi(x_m), y_m))$.
- We train a linear predictor h over $\hat{S}$.
- We predict the label of each test point x as $h(\phi(x))$.

Obviously the success of this learning method will depends on the choice of ϕ: a good ϕ is a function that makes $\hat{S}$ linearly separable in $\mathcal{F}$. The problem behind this procedure is that computing linear separators over very high dimensional data may be computationally expensive. The solution to this concern is kernel based learning.

Definition 5. *Given an embedding $\phi : X \to \mathcal{F}$, we define the kernel function as*

$$K(x, x') =< \phi(x), \phi(x') > . \tag{2.35}$$

We can think of K as a similarity between instances and of the embedding ϕ as mapping X into a space where these similarities are realized as inner products.
Of course not all functions of the form $K : X \times X \to \mathbb{R}$ are kernel function. In fact, k must represent an inner product between $phi(x)$ and $\phi(x')$ for some ϕ. A necessary and sufficient condition for k to be a valid kernel function is given by the *Mercer Theorem*.

Theorem 4. *A symmetric function $K : X \times X \to \mathbb{R}$ implements an inner product in some Hilbert space if and only if it is positive semidefinite; namely for all $x_1, ..., x_m$, the matrix $G_{i,j} := K(x_i, x_j)$, is a positive semidefinite matrix.*

The matrix $G \in \mathbb{R}^{m \times m}$ is called *Gram matrix*. This is useful because we can implement linear separators just on the basis of the values of the kernel function over pairs of points in X, without express ϕ explicitly. Now we will see how.
It is possible to show that each version of SVM optimization problem can be rewritten as

$$\min_{w}(f(< w, \phi(x_i) >, ..., < w, \phi(x_m) >)) + R(||w||) \tag{2.36}$$

where $f : \mathbb{R}^m \to \mathbb{R}$ is an arbitrary function and $R : \mathbb{R}_+ \to \mathbb{R}$ is a monotonically nondecreasing function. The following theorem (*Representer Theorem*), shows that an optimal solution of 2.36 is a linear combination of $\phi(x_1), ..., \phi(x_m)$.

Theorem 5. *Assume that ϕ is a mapping from X to a Hilbert space. Then, there exists a vector $\alpha \in \mathbb{R}^m$ such that $w = \sum_{i=1}^{m} \alpha_i \phi(x_i)$ is an optimal solution of 2.36.*

According to this theorem, we can rewrite $< w, \phi(x_i) >$ as

$$< w, \phi(x_i) >= \Big\langle \sum_j \alpha_j \phi(x_j), \phi(x_i) \Big\rangle = \sum_{j=1}^{m} \alpha_j < w(x_j), \phi(x_i) > \tag{2.37}$$

and $||w||^2$ as

$$||w||^2 = \Big\langle \sum_j \alpha_j \phi(x_j), \sum_j \alpha_j \phi(x_j) \Big\rangle = \sum_{i,j=1}^{m} \alpha_i \alpha_j < \phi(x_i), \phi(x_j) > . \tag{2.38}$$

Therefore problem 2.36 can be rewritten as

$$\min_{\alpha \in \mathbb{R}^m} f\left(\sum_{j=1}^{m} \alpha_j K(x_j, x_1), ..., \sum_{j=1}^{m} \alpha_j K(x_j, x_m) \right) + R\left(\sqrt{\sum_{i,j=1}^{m} \alpha_i \alpha_j K(x_j, x_i)} \right). \tag{2.39}$$

In this equivalent form the mapping ϕ does not appear. The only thing we need to know is the value of the *Gram matrix* G .
Problems in form 2.39 can be solved efficiently and, once we learn α, we can calculate the predictions on a new instance x as

$$< w, \phi(x) >= \sum_{j=1}^{m} \alpha_j < \phi(x_j), \phi(x) >= \sum_{j=1}^{m} \alpha_j K(x_j, x).$$

The advantage of working with kernels is that in some situations the dimension of the feature space is extremely large, while working with kernel function is very simple.
Two of the most commonly used kernels are *Gaussian kernel* and *polynomial kernel*.

- The *Gaussian kernel*, also called *RBF (radial basis functions) kernel*, is defined as
$$K(x, x') = e^{-\frac{||x-x'||^2}{2\sigma}} \quad \forall x, x' \in \mathbb{R}^n. \tag{2.40}$$
This kernel sets the inner products between x and x' in $\mathcal{F}$ to be close to zero if the instances are far away from each other in X and close to 1 if they are close. The parameter $\sigma > 0$ controls the scale determining what we mean by 'close'.

- The *polynomial kernel* of degree k is defined as
$$K(x, x') = (1+ < x, x' >)^k. \tag{2.41}$$
Learning halfspace with a k degree polynomial kernel enables us to learn polynomial predictors of degree k over X.

2.7 K-Nearest Neighbors

In theory we would always like to predict qualitative responses using the Bayer classifier already defined in 2.8. The problem is that, in reality, we do not know the conditional distribution of Y given X and this makes it impossible to apply the Bayes classifier. One very simple approach which can often produce classifiers that are extremely close to the optimal Bayes classifier is the *K-nearest neighbors* (KNN) classifier.
Let us assume that our input space X is endowed with a metric function $\rho : X \times X \to \mathbb{R}$. This metrics return the distance between any two elements of X.
Some of the most common choices for ρ are:

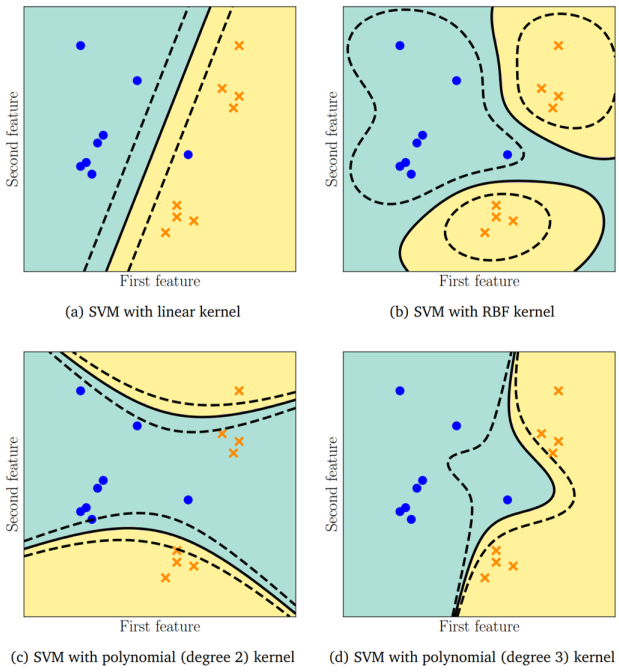

Figure 2.4. SVM with different kernels. The decision boundary is non-linear but the underlying problem is for a linear separating hyperplane.

- the *Euclidean distance*

$$\rho(x, x') = ||x - x'||_2 = \sqrt{\sum_{i=1}^{n}(x_i - x_i')^2} \tag{2.42}$$

- the *Manhattan distance*

$$\rho(x, x') = ||x - x'||_1 = \sqrt{\sum_{i=1}^{n}|x_i - x_i'|} \tag{2.43}$$

- the *Chebyshev distance*

$$\rho(x, x') = ||x - x'||_\infty = \max_{i=1,\ldots,n}|x_i - x_i'| \tag{2.44}$$

- the *Minkowski distance*

$$\rho(x, x') = ||x - x'||_p = \left(\sum_{i=1}^{n}(x_i - x_i')^p\right)^{1/p} \tag{2.45}$$

Given the set S of training samples, for each $x \in X$, let $\pi_1(x), ..., \pi_m(x)$ be a reordering of $\{1, ..., m\}$ according to their distance $\rho(x, x_i)$ to x. This means that, for all $i < m$, this inequality holds:

$$\rho(x, x_{\pi_i(x)} \leq \rho(x, x_{\pi_{i+1}(x)})). \tag{2.46}$$

Given a positive integer K, for all $x \in X$ the KNN classifier first identifies the K points in S that are closest to x. It then approximate the conditional probability $P(Y = y|X = x)$ as

$$P(Y = y|X = x) = \frac{1}{K} \sum_{i \leq K} 1_{y_{\pi_i(x)} = y}. \tag{2.47}$$

Finally, the classifier applies Bayes rule and assign each observation x to the class with the largest probability.

The choice of parameter K has a drastic effect on the classifier. When $K = 1$ the decision boundary is overly flexible and the classifier has a low bias but very high variance. As K increases, the method becomes less flexible and produces a decision boundary closer to linear. This corresponds to a low-variance but high-bias classifier.

It can be proved that the expected error of the KNN rule converges to $(1 + \sqrt{8/K})$ times the error of the Bayes classifier. This is formalized in the following theorem.

Theorem 6. *Let $X = [0,1]^n, Y = \{0,1\}$ and D be a distribution over $X \times Y$ for which the conditional probability function is a c-Lipschitz function. Let h_S denote the result of applying the KNN rule to a sample $S \sim D^m$, where $K \geq 10$. Let h^* be the Bayes optimal hypothesis. Then,*

$$E_S[R(h_S)] \leq \left(1 + \sqrt{\frac{8}{K}}\right) R(h^*) + (6c\sqrt{n} + K)m^{-1/(n+1)}.$$

2.8 Decision Trees

Decision trees are simple, typically fast and very easy to interpret, however they also present some issues. First, their interpretation should be carried

out with care since they suffer from high variance. This means that they are unstable and a small change in the training set could lead to very different trees. Second,they usually are not competitive with other supervised learning methods in terms of prediction accuracy. For this reason, we will later introduce some methods, based on the combination of a large number of trees. We will see that combining trees together leads to a dramatic improvements in prediction accuracy, but costs in terms of loss in interpretation.

A binary decision tree is a tree representation of a partition of the feature

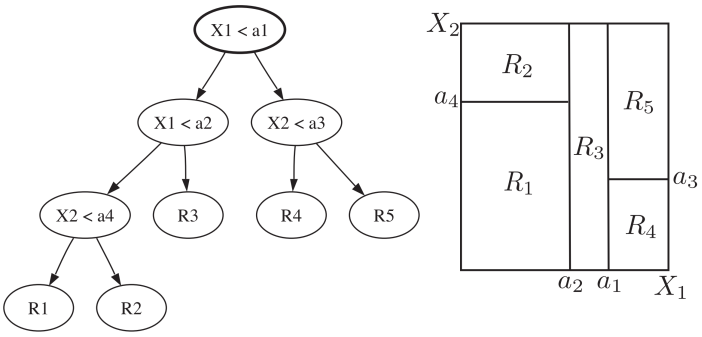

Figure 2.5. Left: scheme of a decision tree with numerical questions based on X_1 and X_2. Right: Partition of the two-dimensional space induced by that tree.

space X. Each interior node of a decision tree corresponds to a question related to features. In case of numerical features, a question will look like $X_i \leq a$, for a certain threshold $a \in \mathbb{R}$. If the variable is categorical, the question will be $X_i \in a_1, a_2, a_3$, where a_1, a_2 and a_3 are possible realization of random variable X_i. Each leaf of the tree is labeled with a label $l \in Y$. From now on we will focus on classification trees.

To predict the label of any point $x \in X$ we start from the root of the tree and go down the tree until a leaf is reached, by moving to the left or to the right according to the response at the question at each node. Once we reached a leaf, we associate the corresponding label to observation x. In other words we partition X in non-overlapping regions. For all point falling

in a certain region we make the same prediction. In classification trees the label of a region is determined using the training sample: the class $\hat{y}$ with the majority representation among the training points falling in the leaf defines the label of that leaf region.
In theory we can partition X in regions of any shape. However, we choose to divide it into high-dimensional rectangles $r_1, ..., r_J$, where J is not fixed a priori. The goal is find $r_1, ..., r_J$ that solve

$$arg \min_{r_1,...,r_J} \sum_{j=1}^{J} \sum_{i \in r_j} L_C(y_i, \hat{y}) \tag{2.48}$$

Unfortunately 2.48 is an NP-hard problem, so it is computationally infeasible.
Instead of solving 2.48, we can learn the tree using two different approach: a top-down greedy technique called *recursive binary splitting* or a *grow-then-prune strategy*.

2.8.1 Recursive binary splitting

In this case we start from a single (root) node, which label is the class that has majority over the train sample. Next, at each round, we split node $\mathbf{n}_t$ based on some question q_t. The pair $(\mathbf{n}_t, q_t)$ is chosen so that the node impurity is maximally decreased according to the measure of impurity F. Let us denote with $F(\mathbf{n})$ the impurity of node $\mathbf{n}$, and with $p_l(\mathbf{n})$ the proportion of training observations at $\mathbf{n}$ that belong to class l. The most commonly used measures F are:

- *misclassification*: $F(\mathbf{n}) = 1 - \max_{l \in Y} p_l(\mathbf{n})$
- *cross-entropy*: $F(\mathbf{n}) = -\sum_{l \in Y} p_l(\mathbf{n}) log_2(p_l(\mathbf{n}))$
- *Gini index*: $F(\mathbf{n}) = \sum_{l \in Y} p_l(\mathbf{n})(1 - p_l(\mathbf{n}))$

Let us denote by with $\mathbf{n}_+(\mathbf{n}, q)$ and $\mathbf{n}_-(\mathbf{n}, q)$ respectively the right and the left child of $\mathbf{n}$ after the split, and with $\eta(\mathbf{n}, q)$ the fraction of points in the

region defined by node $\mathbf{n}$ that are moved to $\mathbf{n}_-(\mathbf{n}, q)$. At each step, we have to maximize the decrease in impurity by solving

$$\max_{(\mathbf{n},q)} F(\mathbf{n}) - [\eta(\mathbf{n}, q)F(\mathbf{n}_-(\mathbf{n}, q)) + (1 - \eta(\mathbf{n}, q))F(\mathbf{n}_+(\mathbf{n}, q))]. \tag{2.49}$$

Gini index and entropy are strictly convex and differentiable, so they are typically the preferred measures to substitute in 2.49. This greedy approach faces some issues. First, a seemingly bad split may leads to subsequent useful splits and so to trees with less impurity overall. Second, to achieve the desired level of impurity, very large trees may be needed and this lead to overfitting.

2.8.2 Grow-then-prune strategy

A better strategy is the so called grow-then-prune strategy. The idea behind this method is to grow a very large tree T_0, and then to prune it back to obtain a subtree $T^* \subset T_0$. This subtree is chosen in order to minimize an objective function defined as the sum of the empirical error and a complexity term that depends on the number of leaves $|T|$of tree T:

$$T^* = arg\min_{\mathbf{n}\in T} |\mathbf{n}|F(\mathbf{n}) + \alpha|T|. \tag{2.50}$$

The parameter $\lambda \geq 0$ is a regularization parameter determining the trade-off between impurity and tree complexity. It is possible to show that the solution T^* to this problem is unique.

In practice, we find T^* by generating a sequence of trees $T_0, T_1, ..., T_n$ starting from the initial tree T_0. For any $i \in [0, n-1]$ we define T_{i+1} from T_i by collapsing one internal node of T_i. This node is chosen so that collapsing it causes the smallest per node increase in the total empirical error of T_i.

2.9 Weak Learning and Ensemble Methods

As we know, it is simpler to find a simple predictor which performs slightly better than random, instead of finding an accurate one which satisfies PAC-learning requirement (see Definition **??**). To better understand what a base

classifier is let us introduce the definition of *weak learning* reported by [8].

Definition 6. *A concept class C is said to be weakly $\gamma-$learnable if there exists an algorithm A, $\gamma > 0$, and a polynomial function $poly(\cdot,\cdot,\cdot,\cdot)$ such that for any $\epsilon > 0$ and $\delta > 0$, for all distribution D on X and for any target concept $c \in C$, the following holds for any sample size $m \geq poly(1/\epsilon,1/\delta,n,size(c))$:*

$$\underset{S\sim D^m}{P}[R(h_S) \leq 1/2 - \gamma] \geq 1 - \delta. \tag{2.51}$$

If such algorithm A exists, it is called a *weak learner* algorithm for C or simply *weak learner*. The hypotheses returned by a weak learner are called *base classifiers*. *Ensemble methods* are methods that combine together several *base classifiers* to create a more accurate one. All these approach are general ones that can be applied to many learning methods both for regression and classification. In this section we will present some of them, focusing on their application on decision trees.

2.9.1 Bagging

Bagging is an ensemble method introduced by Breiman (1996).
As we anticipated before, trees suffer from high variance. However we know that, give a set of n independent observations $Z_1, ..., Z_n$, each with variance σ^2, the variance of the mean $\bar{Z}$ of the observations is given by σ^2/n. Hence,variance can be reduced averaging a set of observation. This is exactly the idea behind bagging.
We can increase prediction accuracy taking many training sets, building a separate prediction model using each of them and then, for a given test observation, we record the class predicted by the majority of B predictors. In practice, we usually do not have access to multiple training set, so we can *bootstrap*, by taking repeated samples from one single training set.

2.9.2 Random forest

The method of random forest was introduced by Breiman (2001). Random forests improve bagged trees by decorrelate them. As in bagging, we build distinct decision trees on bootstrapped training samples. The difference is that at each split of a tree a random sample of p predictors is chosen as split candidates from the full set of n predictors. As a consequence, the split is allowed to use only one of those p predictors. The number p is usually taken as $p = \sqrt{n}$.

This technique can appear strange but it is very effective. Let us suppose to have one very strong predictor in our data set. Then, almost all trees will use this predictor in the first split. Consequently all the trees will look quite similar to each other and will be strongly correlated. If we force each split to consider only p predictors, the probability that the splits will not consider the strong predictor is $(n - p)/n$. Of course, if we choose p to be very similar to n, the performance of the method will be almost identical to bagging. The choice of a very small p is instead very helpful when predictors are strongly correlated. **aggiungere parametri importanti**

2.9.3 Boosting

Boosting is an ensemble method which consist of iteratively learning weak classifiers with respect to a distribution and adding them to a final strong classifier. So, exactly as bagging, boosting use a huge number of trees combined together to obtain a more accurate predictor. Therefore, while in bagging each tree is built on a bootstrapped data set and is independent from all the others, in bootstrap each tree is built on a modified version of the original training set.

The idea behind the method is to improve decision trees prediction by learning slowly, instead of fitting a unique large decision tree, which may lead to overfitting. At each iteration N, a new weak learner $h_i(x)$ is added to the ensemble output $\hat{h}(x)$ according to a specific weight w_i based on the

weak learners accuracy:

$$\hat{h}(x) = \sum_{i=1}^{N} w_i h_i(x). \tag{2.52}$$

Then, the data weights are readjusted: misclassified input data gain a higher weight and examples that are classified correctly lose weight. In this way, future weak learners will focus more on the examples that previous weak learners misclassified.
The choice of the number B of used trees is fundamental: if B is too large boosting can overfit, even if overfitting tends to occur slowly if at all. Another important parameter is the number d of splits in each tree. We often choose $d = 1$ to reduce tree complexity. In this case, we obtain *stumps*, trees consisting of a single split.

2.9.4 AdaBoost

One of the most successful boosting algorithms is *Adaboost* (*Adaptive boost*). AdaBoost uses a set H of base classifiers, which are functions $h : X \to Y$. The family of base classifiers typically used with AdaBoost is that of decision trees. More precisely, stumps which are threshold functions associated to a single feature. In other words, stump correspond to single axis-aligned partitions of the space $\mathbb{R}^n$. As the previous algorithms, AdaBoost take in input the training set S and maintains a distribution D_t over the indices $1, ..., m$. Let us see more in details how it works.

- We set the initial distribution to be uniform

  $$D_1(i) = 1/m \tag{2.53}$$

  for all $i = 1, ..., m$.

- At each round of boosting, that is each iteration $t \in [1, T]$,

  – We choose a new base classifier $h_t \in H$ by minimizing the error ϵ_t

on the training sample weighted by the distribution D_t:

$$h_t \in arg\min_{h \in H} \underset{D_t}{P}[h_t(x_i) \neq y_i] = argmin_{h \in H} \sum_{i=1}^{m} D_t(i) 1_{h(x_i) \neq y_i} \tag{2.54}$$

- We define the coefficient α_t as $\alpha_t = 1/2 log \frac{1-\epsilon_t}{\epsilon_t}$, which is greater than zero if and only if the error ϵ_t is less than $1/2$
- We calculate the normalization factor $Z_t = 2[\epsilon_t(1-\epsilon_t)]^{1/2}$, to ensure that the weights $D_{t+1}(i)$ sum to one.
- For $i = 1, ..., m$ we update the distribution

$$D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_i h_t(x_i))}{Z_t}. \tag{2.55}$$

In this way, the new distribution D_{t+1} is defined from the previous one by increasing the weight on i if x_i is incorrectly classified and decreasing it if x_i is correctly classified.

- After T iterations, the classifier h returned is based on the sign of g, which is a linear combination of the base classifiers h_t

$$g = \sum_{t=1}^{T} \alpha_t h_t, \qquad h = sgn(g). \tag{2.56}$$

The weight α_t assigned to h_t in the linear combination is a logarithmic function of the ratio of the accuracy $1 - \epsilon_t$ and error ϵ_t of h_t. Consequently, more accurate base classifiers are assigned a larger weight in that sum.

It is possible to prove that the empirical error of AdaBoost decreases exponentially fast as a function of T. More precisely:

Theorem 7. *The empirical error of AdaBoost classifier satisfies*

$$\hat{R}(h) \leq exp\Big\{ -2 \sum_{t=1}^{T} \Big(\frac{1}{2} - \epsilon_t\Big)^2 \Big\}. \tag{2.57}$$

Furthermore, if $\gamma \leq (1/2 - \epsilon_t)$ *for all* $t \in [1, T]$*, then*

$$\hat{R}(h) \leq exp(-2\gamma^2 T). \tag{2.58}$$

Note that the name *adaptive boosting* is due to the fact that the algorithm does not need to know the value of γ and the accuracy of the base classifiers, but it adapts to their accuracy and defines a solution based on their values.

2.10 Performance Measures

In the case of binary classification, the most common way to evaluate machine learning algorithms performances is by the so called *confusion matrix*. Its structure is represented in Table 2.1: on the columns we have the pre-

	Predicted Negative	Predicted Positive
Actual Negative	TN	FP
Actual Positive	FN	TP

Table 2.1. Confusion matrix.

dicted classes, while the actual classes stay on the rows. The entries of the matrix are four:

- TN (true negatives) is the number of negative examples correctly classified,
- FP (false positives) is the number of negative examples incorrectly classified as positive,
- FN (false negatives) is the number of positive examples incorrectly classified as negative,
- TP (true positives) is the number of positive examples correctly classified.

The performance measure generically associated to machine learning algorithms is *accuracy*

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}. \tag{2.59}$$

However, in case of unbalanced data set, this is not the best performance metric to use. In fact, in real data sets, it is likely to have a very small number of data point in the positive class ($Y = 1$) compared to that of the negative class ($Y = 0$). In this case an high accuracy do not imply a satisfactory prediction for the minority class, which is the most 'interesting' to predict. Since accuracy is not a good metric for skewed data sets, it is better to evaluate classifiers performances using *recall* (or *sensitivity*) and *precision* (or *positive predictive value*):

$$recall = \frac{TP}{TP + FN}, \qquad precision = \frac{TP}{TP + FP}. \tag{2.60}$$

These two measures can be combined together to obtain the *f measure* (or *F-score*):

$$F = \frac{2 \times precision \times recall}{precision + recall} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}. \tag{2.61}$$

The highest possible value of an F-score is 1.0, indicating perfect precision and recall, and the lowest possible value is 0, if either the precision or the recall is zero.

Another way to evaluate the performance of a statistical model that divide subjects into two classes is *ROC curve* analysis. In order to define the *ROC curve*, we need to introduce two more quantities which are *specificity* and *false positive rate*. The specificity, or true negative rate (TNR), is the conditional probability of correctly classifying the negative examples

$$specificity = \frac{TN}{TN + FP}. \tag{2.62}$$

The false positive rate is the probability of type I error

$$FPR = \frac{FP}{FP + TN} = 1 - specificity. \tag{2.63}$$

Many classification methods give, as output, a quantitative result T (for instance, in our context, the probability of default). In this case it is necessary to define a threshold value c to distinguish the two classes: if $T \geq c$ this example will belong to default class, if $T < c$ this example will be classified as non-default individual. Of course, different values of c lead to different confusion matrix. In particular, as c decreases, sensitivity increases but specificity decreases and viceversa when c increases. The ROC curve illustrates the performance of a binary classification algorithm as its discrimination threshold is varied. The ROC curve can be constructed as a plot of sensitivity versus False Positive Rate, by considering all possible values of c. In particular, two alternative definitions for TPR and FPR can be introduced, making explicit their dependence on c:

$$TPR(c) = P(T \geq c \,|\, Y = 1), \qquad FPR(c) = P(T \geq c \,|\, Y = 0)$$

. The ROC curve is

$$ROC(\cdot) = \{FPR(c), TPR(c),\ c \in (-\infty, +\infty)\} \tag{2.64}$$

or, equivalently,

$$ROC(\cdot) = \{(t, ROC(t)),\ \ t \in (0,1)\} \tag{2.65}$$

where the ROC function maps t to $TPR(c)$, and c is the threshold corresponding to $FPR(c) = t$.
An uninformative method is one such that $TPR(c) = FPR(c)$ for every threshold C and this is represented by a line with unit slope (a ROc curve $ROC(t) = t$). An ideal classification algorithm completely separates defaults from non-defaults cases, i.e. $TPR(c) = 1$ and $FPR(c) = 1$ for some C. This means that the ROC curve is along the left and upper borders of the positive unit quadrant.

The most important numerical index used to describe the behavior of the ROC curve is the *area under the ROC curve* (AUC), defines as follows:

$$AUC = \int_0^1 ROC(t)dt \tag{2.66}$$

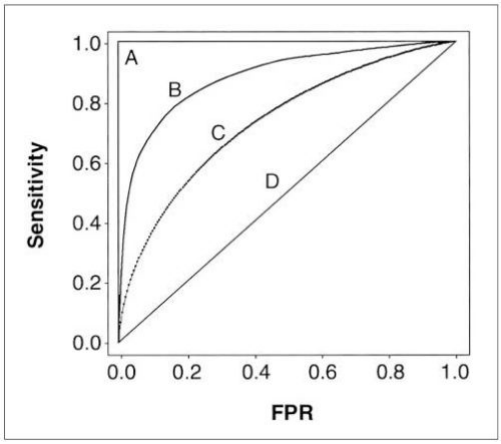

Figure 2.6. ROC curve example for four different models. Model D is completely uninformative, while model A represent the idel one.

A hypothetical ideal model would have $AUC = 1$. As the test accuracy decreases, the ROC curve moves toward the diagonal line corresponding to uninformative model with $AUC = 0.5$.

2.11 Unsupervised Learning

As anticipated before, the fundamental difference between supervised and unsupervised learning is whether the examples given to the learning algorithm are labeled or not. Unsupervised machine learning algorithms are applied to unlabeled examples, and in most cases, particularly referring to clustering algorithms, are used as a data mining technique to cluster examples into groups of similar objects instead of giving predictions directly. Unsupervised algorithm are extensively applied in credit risk assessment as complementary tools to supervised ones.

2.12 Clustering

This section is developed with reference to [4] and [14].
Clustering refers to a set of techniques for finding subgroups, or *clusters*, in

a data set. More precisely, clustering is the task of partitioning a data set in such a way that the observations within each group are quite similar to each other, while observations in different groups are quite different from each other.

However, there are several sources for this difficulties. The first one is that the two objective mentioned in the earlier statement may contradict each other. From a mathematical point of view, similarity (or proximity) is not a transitive relation, while cluster sharing is an equivalence relation and, in particular, a transitive one. To better understand this concept, a simple example can be introduced. Let us suppose to have a long sequence of objects $x_1, ..., x_m$ such that each x_i is very similar to its two neighbors x_{i-1} and x_{i+1}, but x_1 and x_m are very dissimilar. If we wish to assure that whenever two elements are similar they share the same cluster, then we must put all the elements of the sequence in the same cluster. However, in this case, we end up with a cluster that also contains dissimilar elements such as x_1 and x_m. Thus we are violating the second requirement.

The second problem is the lack of "ground truth", which is a common problem in unsupervised learning. In fact, while a supervised learner can estimate the success of its hypotheses using the labeled training data by computing the empirical loss, in supervised learning there are no labels that we try to predict, we only wish to organize the data in some meaningful way. As a result, there is no clear success evaluation procedure for unsupervised algorithm. The main consequence of these two problems is that a given data set can be clustered in various different meaningful ways. As a result, there is a wide variety of clustering algorithms that, if applied on the same input data, will output very different clusterings.

Clustering algorithms can vary in terms of both the type of input and the type of outcome they are expected to compute. In general, a cluster algorithm receives in input a set X of elements and a distance function ρ over it. The function $\rho : X \times X \to \mathbf{R}_+$ is symmetric and satisfies both $\rho(x, x) = 0$ for all $x \in X$ and the triangle inequality. The most popular

distance metrics are Euclidean distance, Manhattan distance, Minkowski distance and Chebyshev distance, defined in 2.42-2.45.
An alternative to ρ could be a similarity function $s : X \times X \to [0,1]$ that is symmetric and satisfies $s(x, x) = 0$ for all $x \in X$. Moreover, some clustering algorithms require as input also a parameter k, which represent the number of required clusters.
The output of a cluster algorithm usually is a partition $C = (C_1, ..., C_k)$ of the domain set X. It means that $\cup_{i=1}^{k} C_i = X$ and for all $i \neq j$, $C_i \cap C_j = \emptyset$.
Another possible output is a clustering *dendrogram*, which is a hierarchical tree of domain subset, having the singleton sets in its leaves, and the full domain as its root.
In the following section we will focus on cost minimization clusterings and, in particular, on $k-$means clustering.

2.12.1 Cost minimization clusterings

One possible approach to clustering could be defining a cost function over a set of possible clusterings and the goal of clustering algorithm is to find the minimal cost partitioning. In such a way the clustering task can be seen as an optimization problem. Its objective function, denoted by G, is a function from pairs of an input, (X, ρ), and a proposed clustering solution $C = (C_1, ..., C_k)$, to positive real numbers. The goal of the algorithm is to find a clustering C such that $G((X, \rho), C)$ is minimized. However, most of the resulting optimization problems are NP-hard, so when we talk about k-means clustering, for example, we refer to some particular approximation algorithm rather than the cost function or the corresponding exact solution of the minimization problem.
Many common objective function requires the number k of clusters as a parameter. The choice of k is left to the user; we will see in next section which are the most common method to make this choice.
One of the most popular objectives is *k-means* objective function. Let us assume that the input space X is embedded in a larger metric space (X', ρ),

so that $X \subseteq X'$. In k-means each C_i is represented by a *centroid* $\mu_i \in X'$ defined as follows:

$$\mu_i(C_i) = arg \min_{\mu \in X'} \sum_{x \in C_i} \rho(x, \mu)^2. \tag{2.67}$$

The k-means objective function measures the squared distance between each point in X and the centroid of its cluster:

$$G_{k-means}((X, \rho), (C_1, ..., C_k)) = \sum_{i=1}^{k} \sum_{x \in C_i} \rho(x, \mu_i(C_i))^2 \tag{2.68}$$

or, equivalently,

$$G_{k-means}((X, \rho), (C_1, ..., C_k)) = \min_{\mu_1, ..., \mu_k \in X'} \sum_{i=1}^{k} \sum_{x \in C-i} \rho(x, \mu_i)^2. \tag{2.69}$$

There exist some common variations of this objective function. For example, one can require the cluster centroids to be members of the input set. In such way we obtain the so called *k-medoids* objective function, which is defined as follows:

$$G_{k-medoid}((X, \rho), (C_1, ..., C_k)) = \min_{\mu_1, ..., \mu_k \in X} \sum_{i=1}^{k} \sum_{x \in C_i} \rho(x, \mu_i)^2. \tag{2.70}$$

Alternatively, one can measure the "distortion" between a data point and the centroid of its cluster by distance rather than by the square of the distance. In this case we get the so called *k-median* objective function

$$G_{k-median}((X, \rho), (C_1, ..., C_k)) = \min_{\mu_1, ..., \mu_k \in X} \sum_{i=1}^{k} \sum_{x \in C_i} \rho(x, \mu_i). \tag{2.71}$$

2.12.2 Elbow and Silhouette methods

Elbow and Silhouette methods are the two state-of-the-art methods used to identify the correct cluster number k in a data set. They both base their functioning on two internal measures, cluster cohesion and cluster separation, which determine how good a clustering is without external information.

The idea behind the *Elbow method* is to chose the correct number of clusters K according to a cohesion measure such as the *within cluster sum of square* (WSS)

$$WSS = \sum_{i=1}^{k} \sum_{x \in C_i} (x - \mu_i)^2. \tag{2.72}$$

The method consist in evaluate the WSS for different values of k, starting from $k = 1$, up to a maximal specified k. Each value of WSS is listened in a graph where the y-axes label is the value of the WSS and the x-axis label is the number of clusters. Elbow method suggests that the best value of K corresponds to the point in which the graph has a significant bend. There is, however, a problem with the Elbow method: the elbow point cannot be unambiguously distinguished when the plotted curve is fairly smooth. The Silhouette method is another well-known method with decent performance to estimate the potential optimal cluster number, which uses the average distance between one data point and others in the same cluster and the average distance among different clusters to score the clustering result. The metric of scoring of this method is named the *silhouette coefficient* S, and defined as

$$S = \max_{x_i \in X} S(x_i) \tag{2.73}$$

where, for each data point x_i in cluster C_I,

$$S(x_i) = \begin{cases} \frac{b(x_i) - a(x_i)}{max\{a(x_i), b(x_i)\}} & \text{se } k > 1 \\ 0 & \text{se } k = 1 \end{cases} \tag{2.74}$$

where

$$a(x_i) = \frac{1}{|C_I| - 1} \sum_{x_j \in C_I, i \neq j} \rho(x_i, x_j) \tag{2.75}$$

and

$$b(x_i) = \min_{J \neq I} \frac{1}{|C_J|} \sum_{x_j \in C_J} \rho(x_i, x_j). \tag{2.76}$$

In other words, a and b represent the mean intra-cluster distance and the mean nearest-cluster distance, respectively.

It is quite clear, from the above definition, that $S \in [-1,1]$. A value of S

closer to 1 indicates that a sample is better clustered, while a value of S closer to -1, indicates that the sample should be categorized into another cluster.
Silhouette method is preferable for estimating the potential optimal cluster number.

Chapter 3

Application on data

3.1 Motivation

This work focuses on integrating unsupervised machine learning techniques with supervised machine learning classifiers to improve performances of credit scoring models. This approach refers to [15].
To test the validity of the integration of unsupervised and supervised machine learning techniques, we compare individual models and cluster-based models. In terms of individual models, we will adopt the set of ML algorithms presented in Chapter 2 (logistic regression (LR), support vector machine (SVM), k-nearest neighborhood (KNN), random forest (RF) and AdaBoost (AB)).
The rest of this chapter is organized as follows: Section 3.2 provides an introduction to the data set, while Section 3.4 presents data cleaning and oversampling and features selection methods.

3.2 Data Set Description

A significant problem for credit scoring models which must be pointed out is the unavailability of real-world credit data. The reason is that customer's credit data is confidential in most of the financial institutions. For these

reasons we will use a public data set from Kaggle.
Kaggle is an online community of data scientists and machine learning pratictioners that offers machine learning competitions and a public data platform.
The data set contains information on clients who have taken out a loan with an unspecified financial institution. The aims of the analysis is to search for statistical relationships that could give us some insights about the the risk of credit and to develop some algorithms to predict credit default. Given that the context of the data set is not fully explained, we are not going to take into account external macroeconomic events, which could completely change the results of the analysis. For instance, different countries could have different loan's requirement, or different phases of the economic cycle could end up in a very different scenario.
Another important fact to which one must draw attention is that we will assume the data set to be not distorted by a credit score system. This means that no systematic screening of the costumers' credit standing had been implemented until the date of data retrieval.
The data set contains 12 attributes and 32581 observations, and its structure is resumed in Table 3.1.

The potential covariates describe both customers and loans characteristics. As we can see there are both numerical and categorical variables. Let us see a little more in details which are the levels of the categorical ones.

- *person_home_hownership* has four levels: Mortage, Own, Rent and Others

- *loan_intent* has six different levels: Debt consolidation, Education, Home improvement, Medical, Personal, Venture

- *loan_grade* has seven levels: A, B, C, D, E, F, G. Th grade takes into account a combination of several indicators of credit risk from the credit report and loan application. These factors may include the level of guarantor support, repayment history, cash flow, projected yearly

VARIABLE	DESCRIPTION	TYPE
person_age	Customers age (in years)	numerical
person_income	Annual income (in dollars)	numerical
person_home_ownership	Home ownership	categorical
person_emp_length	Employment length (in years)	numerical
loan_intent	Loan intent	categorical
loan_grade	Loan grade	categorical
loan_amnt	Loan amount (in dollars)	numerical
loan_int_rate	Loan interest rate	numerical
loan_status	Loan status	categorical
loan_percent_income	Loan percentage income	numerical
cb_person_default_on_file	Historical default (Y or N)	categorical
cb_person_cred_hist_length	Credit history length (in years)	numerical

Table 3.1. Dataset description

expenses, etc.

- *cb_person_default_on_file* has only two level: *Y* if the client has already had a default, *N* otherwise

- *loan_status* is the response variable and has two levels: 0 represent non default, 1 represent default.

3.3 Data Exploration

In this section we will use data visualization to have a general view on the data set and to get an idea of what are the most relevant features for detecting a default.
First of all, we report in Figure 3.1 the frequency graph of the two classes (default and non-default) we are analyzing. It is quite clear that the data

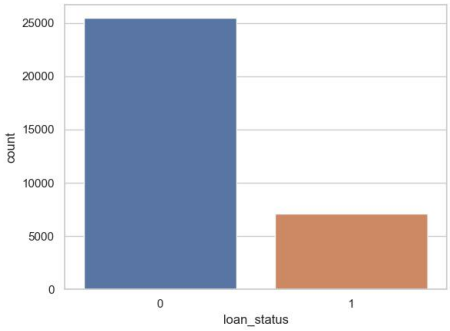

Figure 3.1. Count of the two classes default=0 and default=1.

set is deeply unbalanced, in fact only the 21,8% of the customer belong to the class 1. This problem will be solved lately, in Section 3.4.4.

In figure 3.2 we can see some plots which represent the proportion of default and non default customers for each class of data set categorical variables. The proportion of default customers, not surprisingly, increases as the the loan grade decreases.

Similarly, the proportion of default customer is higher (almost 40%) for the ones that have already registered one or more defaults in their history, while it is smaller (less than 20%) for those who have never had a default. regarding home ownership, the proportion of defaults is higher for people who are renting a house, while it is slightly higher than 10% for those who are paying a mortgage. The class of *person_home_ownership* with the smallest proportion of defaults is the one of those who live in their own house.

The proportion of defaults is almost the same for loan intents such as educational, venture and personal. It is slightly higher for medical and home improvement intent and even higher for debt consolidation.

Figure 3.3 shows box plots for the quantitative variable of the data set. These allow us to start getting an idea of what the outliers are (we will better analyze them in section 3.4.1), and allow us to detect some important differences between numerical attribute distribution for class 0 and for

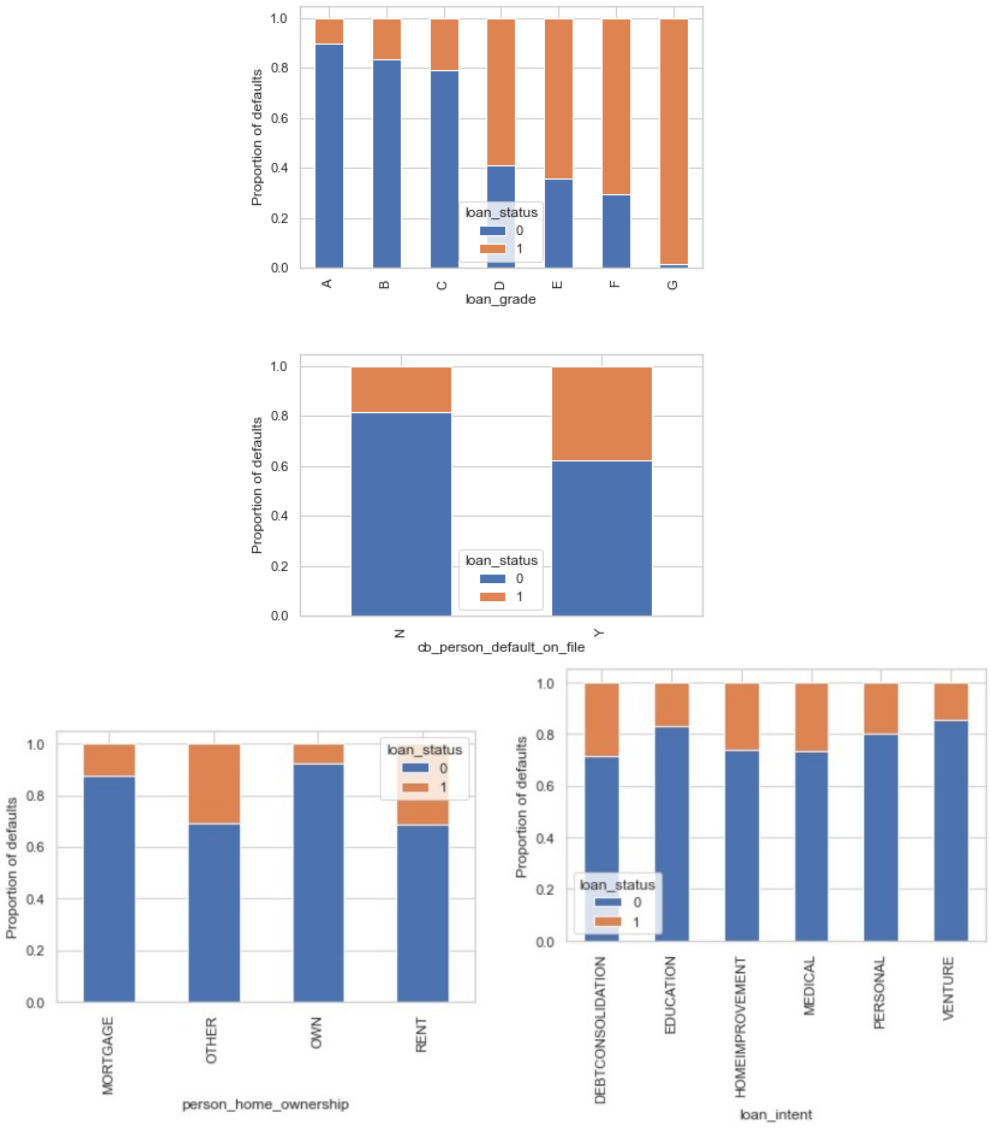

Figure 3.2. Proportion of default and non default individuals for different classes of categorical variables.

class 1.

The first one, the box plot regarding customer age, do not shows any significant difference in customer age between the two classes, but it highlights the presence of outliers in the data set. In fact, there are several point

indicating customer which are more than 120 years old.
The box plot regarding customers income shows, once again, the presence of outliers in the data. Furthermore, it shows that the average income of non default people is higher than the mean income of default customers (more precisely 70804 dollars versus 49125).
Employment length mean is slightly smaller for people classified as default customer (approximately 4.14 years), with respect to the ones classified as non default customers (approximately 4.97 years).
Loan amount is basically higher for default customers (10850 dollars versus 9237 dollars for non default customer).
The biggest differences between the two classes are given by their mean loan interest rate and by the mean loan percentage income. Average loan interest rate is approximately 10.43 for non default customers and 13.06 for the default customers. Mean loan percentage income is, again, smaller for class 0 (approximately 0.1488) and higher for class 1 (approximately 0.2468).
The last box plot, the one about credit history length, is not particularly significant, since the two distributions are almost identical. Figure 3.4 shows a correlation heatmap involving all numerical features of the data set, plus the response variable. We observe that the highest correlation is the one between *person_age* and *cb_person_cred_hist_length*. These two variables are positive correlated with correlation 0.86. This is quite reasonable, since older people will probably have a longer credit history. Another, even if less strongly, correlated couple of features is the one composed by *loan_amnt* and *loan_percent_income*. Their correlation is 0.57. All other numerical potential predictors have a correlation, in absolute value, smaller than 0.27. For all these reason we will only consider as redundant the two couples of predictors mentioned above. This means that in the phase of feature selection (see Section 3.4.5), we will make sure that two features from the same couple will not be selected at the same time.

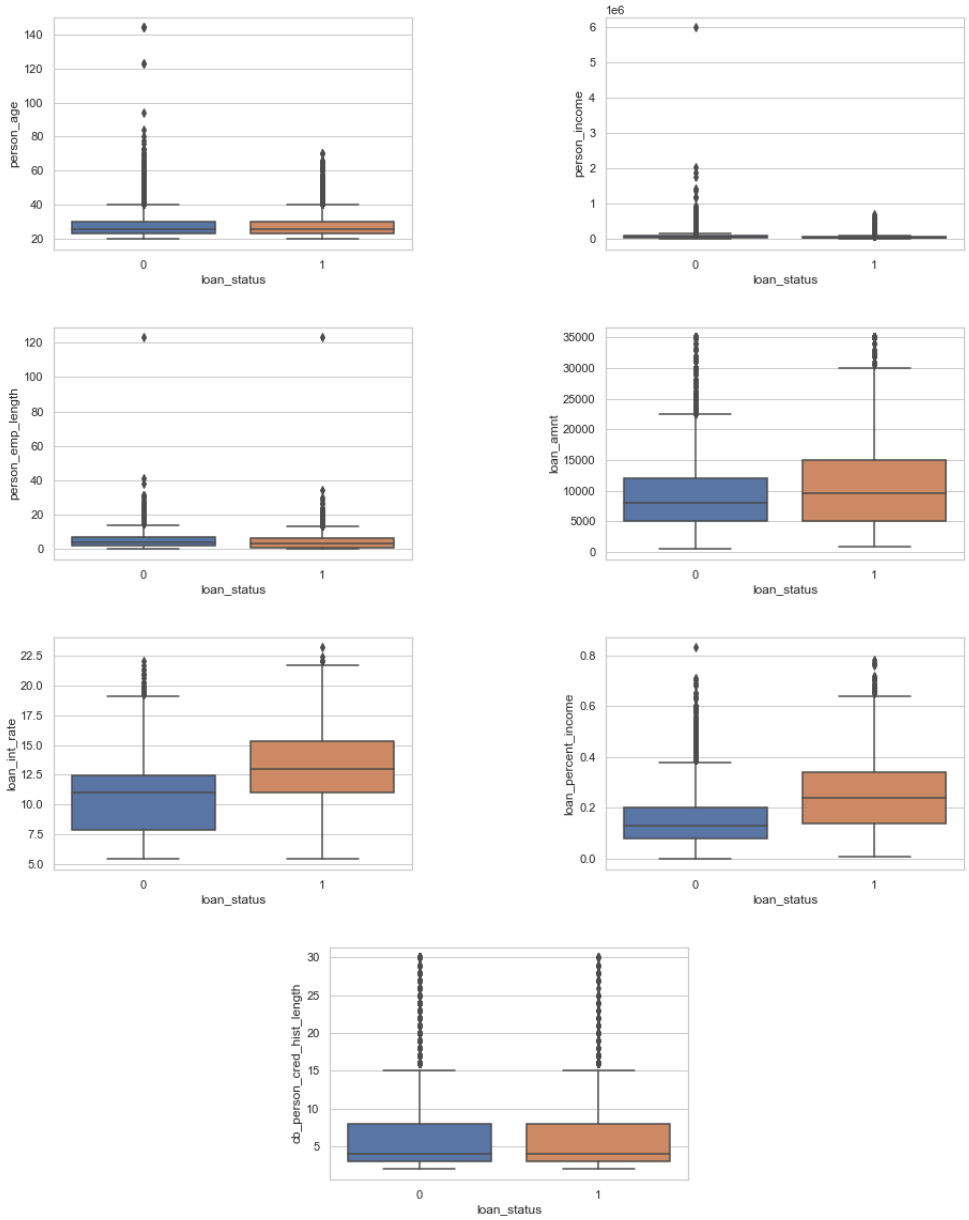

Figure 3.3. Box plots for quantitative variables.

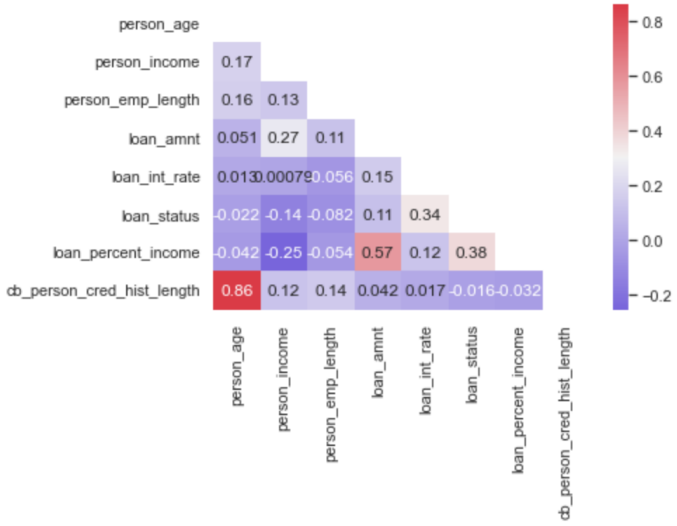

Figure 3.4. Correlation heatmap.

3.4 Data Preprocessing

3.4.1 Missing values

The data set contains some missing values in two variables: *person_emp_length* and *loan_int_rate*. Missing values can be replaced in many different ways. In case of a numerical variable it is common to replace them by a standard value, the mean or the median. However, the most used one is the median, since the mean suffers from the presence of outliers. In case of categorical variables, the missing values are usually replaced by mode or by adding a new category which indicate the missing value. In our case, we decide to replace *loan_int_rate* with the median and *person_emp_length* with the mode. This decision is due to the fact that *person_emp_length* is a quantitative attribute which only assume integer values, for this reason it can be, possibly, considered as a qualitative one with a very large number of labels.

VARIABLE	% MISSING VALUES
person_age	0%
person_income	0%
person_home_ownership	0%
person_emp_length	2.75%
loan_intent	0%
loan_grade	0%
loan_amnt	0%
loan_int_rate	9.56%
loan_status	0%
loan_percent_income	0%
cb_person_default_on_file	0%
cb_person_cred_hist_length	0%

Table 3.2. Percentage of missing values for each feature.

3.4.2 Outliers detection

As anticipated in Section 3.3, the data set contains a certain number of outliers. Figure 3.5, although not exhaustively, give an idea of how we detect them. In particular, we decide to eliminate from the data set all the observations that present one or more of the following characteristic:

- customer age greater than 110 years,
- employment length greater than 100 years,
- employment length greater than customer age,
- annual income greater than $5 \cdot 10^6$ dollars.

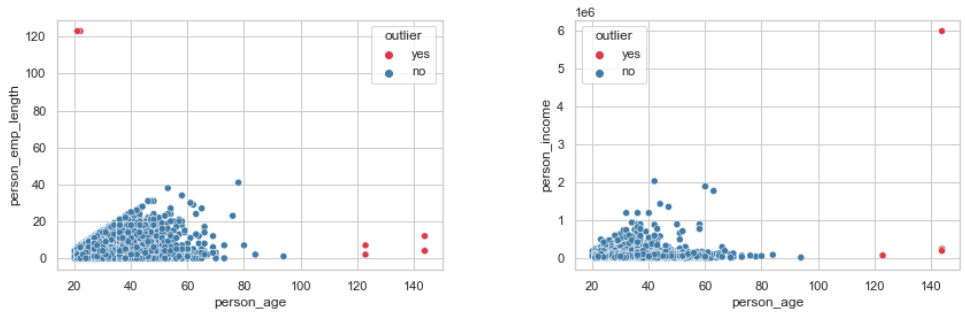

Figure 3.5. Data set outliers (in red).

3.4.3 Dataset encoding

Before proceeding, we transform categorical variables using *get_dummies* method. In this way we obtain a dummy variable for each level of a categorical one. Dummy variables are binary variables which take value 1 if a certain categorical variable take on a specific value, 0 otherwise.
After this passage, the data set contains 27 variables (including the response one).

3.4.4 Oversampling with SMOTE

Real world data sets are often composed of 'normal' examples and, only in small percentage, of 'abnormal' examples which are usually the ones interesting for the analysis. In fact, the cost of misclassifying an 'abnormal' example is often much higher than the cost of the reverse error.
This is exactly the case of our data set. Figure 3.1 shows that the data set is deeply unbalanced: 25473 observations belong to class 0 and only 7108 people belong to class 1. The imbalance in the data set causes problems 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. Therefore, it is necessary to apply a sampling technique. In this case, after splitting the data set into train and test sets (respectively 70% and 30% of the initial data set), we choose to oversample the test set

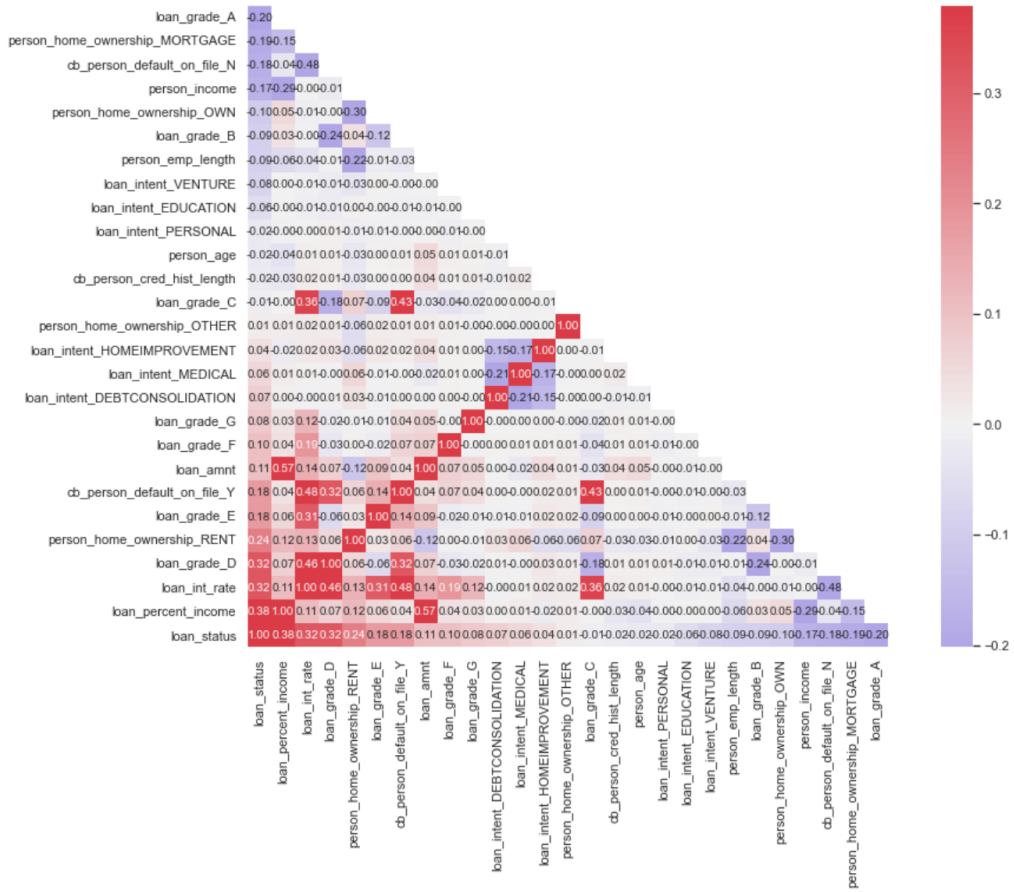

Figure 3.6. Correlation heatmap for encoded data.

using SMOTE (*synthetic minority oversampling technique*). This approach consists in creating extra training data in the minority class by taking each minority class sample and introducing synthetic examples along the line joining any of the k minority class nearest neighbors. Neighbors form the k nearest neighbors are randomly chosen depending on the amount of oversampling we want to perform. Synthetic samples are generated by taking the difference between the feature sample and its neighbor. Once we get this difference, we multiply it for a random number in [0,1] and we add the result to the feature vector we started from. In other words, we select a point on the line segment which connect two specific examples (as shown

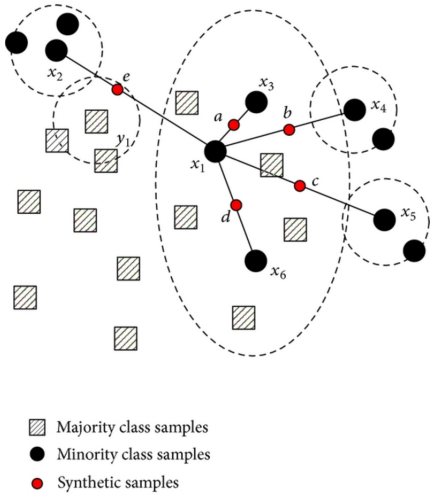

Figure 3.7. Example of SMOTE procedure.

in Figure 3.7). The synthetic examples cause the ML classifiers learn for the minority class sample instead of being subsumed by the majority class samples around them. This results in algorithms that generalize better.
By applying this technique to the train set, we get a perfectly balanced train set consisting of 35542 observation, half of which belong to class 0 and the remaining half to class 1.

3.4.5 Features selection

For our analysis we will use both the complete data set and a new one composed only by a reduced number of features obtained using RFECV (*recursive feature elimination with cross validation*). This is a recursive feature elimination with automatic tuning of the number of features selected with cross-validation. The goal of recursive feature elimination is to select significant features by recursively considering smaller and smaller sets of features.First, an external estimator is trained on the initial set of features and the importance of each feature is calculated. Secondly, the least important feature is pruned. This procedure is repeated iteratively

until we reached the optimal number of features.
By applying this algorithm to the encoded data set we get that the optimal number of features is 15.

Chapter 4

Models Settings and Performances

In this chapter we will analyze in details all methods settings and performances. We will first introduce individual models hyperparameters tuning and results, then we will proceed with cluster-based models ones. At the end, in Section 4.4, we will comment performances also from an economic point of view.
All the supervised and unsupervised models used in this work are completed using Scikitlearn library.

4.1 Individual models

Let us start from individual models. Firstly, we apply all individual models to the entire data set and then to the data set containing only the features selected using RFECV. In both cases, except for logistic regression, we have to optimize the hyper-parameters of each algorithm. For SVM and KNN we use a simple grid search, while for RF and AB we use a randomized search within 5-cross-validation followed by a grid search where the grid is based on the results of randomized search.

Since grid search uses an exhaustive search of pre-defined hyper-parameter space, we now provide the search space for SVM and KNN algorithms. For SVM model we try to adopt both a linear and a RBF kernel. The two parameters to tune are C and γ: C trades off correct classification of training examples against maximization of the decision function's margin, γ is the inverse of the radius of influence of samples selected by the model as support vectors. In particular, we choose C is in range of $(1, 10, 100, 1000)$ and, in the case of RBF kernel (see 2.40), we choose γ in the range of $(0.0001, 0.001)$. For both complete and feature selected data set, we get that the best hyper-parameters combination is a RBF kernel with $C = 1000, \gamma = 0.001$. For KNN model the number of neighbors K is searched in the range of 5 to 50. For complete data set the optimal K is 5, for feature selected data set it is 9.
Regarding RF, the randomized search is based on the following grid: the maximum depth of the tree is in the range of $(10, 33, 56, 80, None)$ (where 'None' include the possibility of not having a predefined maximum depth), the number of features considered at each split is in range of $('auto', 'sqrt')$, the minimum number of observations at each split is in the range of $(2, 5, 10)$ and the number of estimators is in the range of $(200, 650, 1100, 1550, 2000)$. The best combination of parameters is different for the two data sets. For the complete one is: number of estimators equal to 200, minimum number of samples at each split and minimum number of samples at each leaf both equal to 2, number of observations considered at each split $'sqrt'$ and maximum depth of each tree equal to 33. For the feature selected data set we get respectively, 1550, 10, 4, $'sqrt'$ and 80. The two grids of the grid search steps are therefore different, in fact each grid is centered on the optimal combination found with the randomized search. For example, in the first case (complete data set) the grid for the second step is defined as follows: the maximum depth of the tree is in the range of $(25, 30, 33, 35)$, the number of features considered at each split is $'sqrt'$, the minimum number of observations at each split is in the range of $(2, 5)$ and the number

of estimators is 200. At the end we get that the optimal results found at the first step is still optimal after the second one (both for complete and incomplete data set).
For AB classifier, we have to tune two parameters: the number of estimators and the learning rate. The randomized search i based on the following grid: the learning rate is in the range of $(0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1)$, while the number of estimators is in the range of $(0, 20, 40, ..., 1000)$. In the case of complete data set we get that the best combination for this first step is the one with learning rate 0.9 and number of estimators 897, so the grid for the grid search step is defined as follows: the number of estimators is in the range of $(850, 897, 900, 950)$ and the learning rate is in the range of $(0.85, 0.88, 0.9, 0.92, 0.95)$. At the end we get that the best hyper-parameters combination is: number of estimators equal to 950 and learning rate equal to 0.92. Proceeding in a similar way we find out that the best hyper-parameters combination in the case of feature selected data set is: number of estimators equal to 102 and learning rate 0.4.
The performance of individual models are resumed in Table 4.1 and 4.2 and are evaluated according to the measures defined in Section 2.10. All the measures are calculated according to a test set which consists of 30% of the initial data set.

Since each evaluation parameter describes a different aspect of model performances, and there is no one single classifier that is preferable to others on all parameters, we can use F-score in order to compare different methods. This choice is due to the fact that F-score considers the confusion matrix more comprehensively than the other parameters and it is more efficient to reflect the model performance of unbalanced data set. We will then focus on AUC, type I and type II errors in Section 4.3.
In the case of complete data set, the best model in terms of F1-score is random forest, with 0.817. This model also reaches the best performances in terms of AUC, accuracy and precision. If we consider the results of individual models applied on the feature selected data set, we note that

Individual models (26 features)					
	LR	SVM	KNN	RF	AB
AUC	0.783	0.892	0.858	**0.925**	0.904
Accuracy	0.692	0.908	0.892	**0.932**	0.890
Recall	**0.732**	0.648	0.616	0.713	0.674
Precision	0.382	0.888	0.833	**0.957**	0.778
F-score	0.502	0.749	0.708	**0.817**	0.722
Type I error	0.319	0.022	0.033	**0.009**	0.052
Type II error	**0.268**	0.352	0.384	0.287	0.326

Table 4.1. Individual models performances on entire data set.

Individual models (15 features)					
	LR	SVM	KNN	RF	AB
AUC	0.849	0.850	0.850	**0.867**	0.858
Accuracy	0.866	**0.894**	0.892	**0.894**	0.880
Recall	0.573	0.623	0.618	0.618	**0.673**
Precision	0.737	0.837	0.833	**0.843**	0.740
F-score	0.645	**0.714**	0.710	0.713	0.705
Type I error	0.055	0.033	0.033	**0.031**	0.064
Type II error	0.427	0.377	0.382	0.382	**0.327**

Table 4.2. Individual models performances on features selected data set.

SVM is the one with higher F-score (0.714), immediately followed by RF (with 0.713).

In general, it is difficult to establish whether the usage of entire data set is better than the feature selected one. The improvement or deterioration of the results depends on the considered method and measure. However we will give priority to the interpretability of the method and we will proceed

with cluster-based methods only on feature selected data set.

4.2 Cluster-based models

We now adopt the same five base learners to build models based on subsets clustered by k-means, in order to explore whether the use of unsupervised clustering on a data set can help improve predictive performance on credit scoring models.
First of all we use both elbow and silhouette methods to determine the most suitable k. In particular, using Python KneeLocator, we find out that the k corresponding to the point in which WSS (defined in 2.72) has a significant bend is $k = 4$. We then check that the second method also agrees with the choice of k. Actually, the point $k = 4$ is a local maximum for silhouette coefficient.
Secondly, we split the feature selected data set into 4 subsets according to k-means algorithm and we split each cluster into train and test set (again, we use a train set which dimension is 30% of the original cluster).
Then, we apply all the five methods to each one of the cluster, repeating every time the procedure used for the individual models to find the optimal combination of hyper-parameters.
The results of cluster-based models performances can be seen in Table 4.3, while the comparison with the performances of individual models is presented graphically in Figure 4.1.

According to F-score values, five cluster-based models (cluster-based LR, cluster-based SVM, cluster-based KNN, cluster-based RF, cluster-based AB) outperform their counterparts built with base learners to different degrees. The clustering method improves the LR by the greatest degree, with the F-score of 0.645 improved to 0.692. Followed by the clustering results based on the AB model, with the F-score of 0.705 improved to 0.734. The F-score values of the other three models (SVM, KNN, RF) are also slightly improved. Among all the cluster-based models, the cluster-based

Cluster-based models (4 clusters on selected features)					
	LR	SVM	KNN	RF	AB
AUC	0.869	0.862	0.848	**0.874**	0.841
Accuracy	0.874	0.889	0.891	**0.896**	0.891
Recall	0.643	0.657	0.629	0.646	**0.688**
Precision	0.748	0.800	0.832	**0.843**	0.787
F-score	0.692	0.721	0.716	0.731	**0.734**
Type I error	0.061	0.043	0.036	**0.034**	0.052
Type II error	0.357	0.343	0.370	0.354	**0.312**

Table 4.3. Cluster-based models performances based on features selected data set.

AB has the best performance with the F-score of 0.734, which is better than the best F-score for individual models (0.714).
From the results above, we can conclude that the clustering method implemented on this data set helps improve the performances of individual models.

4.3 ROC Curve Analysis

ROC curve and AUC comparison between different classifiers is represented in Figure 4.2. The usage of the feature selected data set strongly improves logistic regression AUC and makes slightly decrease all others AUC.
Let us now focus on the comparison between ROC curves of individual models and cluster-based models (both on feature selected data set). It can be observed that RF model achieves the highest AUC among the individual models and cluster-based models (AUC values are respectively 0.867 and 0.870). The fact that RF performs quite well in terms of AUC means that RF models exhibit a good level of capability on ranking the default against non-defaults and on achieving balance between the type I and type II error.

Figure 4.1. Comparison between performance measures for individual models (in blue) and for cluster-based models (in orange) on feature selected data set.

In most cases (LR, SVM, RF), performances of the cluster-based models are better than those of individual models, which is consistent with the previous conclusion drawn from other measuring parameters (such as F1-score) that data set clustering can help to improve the predictive performance.

4.4 Expected Misclassification Cost

Note that AUC is not the only parameter to evaluate ROC curves. It is possible to observe that there are some intersects among curves of different models. This means that model performance varies according to the cut-off

thresholds, which is related to the ratio between type I and type II error. ROC curve is, in fact, a good indicator of the balance between type I and type II error which are two key parameters with a great effect on the profitability of financial institutions. In fact, as written in [9], the real world financial failure prediction problem is deeply characterized by cost sensitivity as a focus of recent study. Cost-sensitivity indicates that the misclassification costs are non-uniform: the cost due to classifying a bad credit as a good one is usually greater than the one due to classifying a good credit as a bad one. As a consequence, the performance of a classifier on the distressed class is more important than the overall performance. One possible solution to this problem is to train a cost-sensitive classifier with the misclassification cost of the 'bad' creditors greater than that of good ones. According to Turney (2000), there are several different categories of misclassification cost: constant error cost of class, conditional error cost dependent on individual instance, time cost, feature cost and test cost. Nowadays, the most used one is the first one, which, in the case of financial failure, can be simply represented in a 2D matrix (see Table 4.4). The

Real class	Predicted class	
	Positive	Negative
Positive	0	C_p
Negative	C_n	0

Table 4.4. A 2D misclassification cost matrix (*positive* bad creditors, *negative* good creditors).

diagonal entries of the matrix are 0, while the off-diagonals ones are C_p and C_n, which respectively denotes the misclassification cost associated with positive class and with negative class. Normally the condition $C_p > C_n$ holds. At this point we can introduce a new measure, the *expected misclassified cost* (EMC), which can be used as for comparison between classification performance. Expected misclassified cost is strictly related to

type I and type II error (or, equivalently, FPR and FNR) by the following relation:

$$EMC = C_p \cdot FNR + C_n \cdot FPR. \tag{4.1}$$

The most commonly used method to determine C_p and C_n is by calibration based on previous studies. For examples, Dr. Hofmann, when dealing with credit data sets, suggests to use as relative costs of misclassification respectively 5 and 1. Figure 4.3 illustrates EMC for all five methods, both for simple and cluster-based ones. The method with lowest EMC value is AdaBoost classifier, immediately followed by support vectors machine classifier and random forest. Furthermore, we can note once again that the usage of unsupervised clustering before applying supervised methods is profitable, in fact it causes a reduction of the expected misclassification cost for all five algorithms.

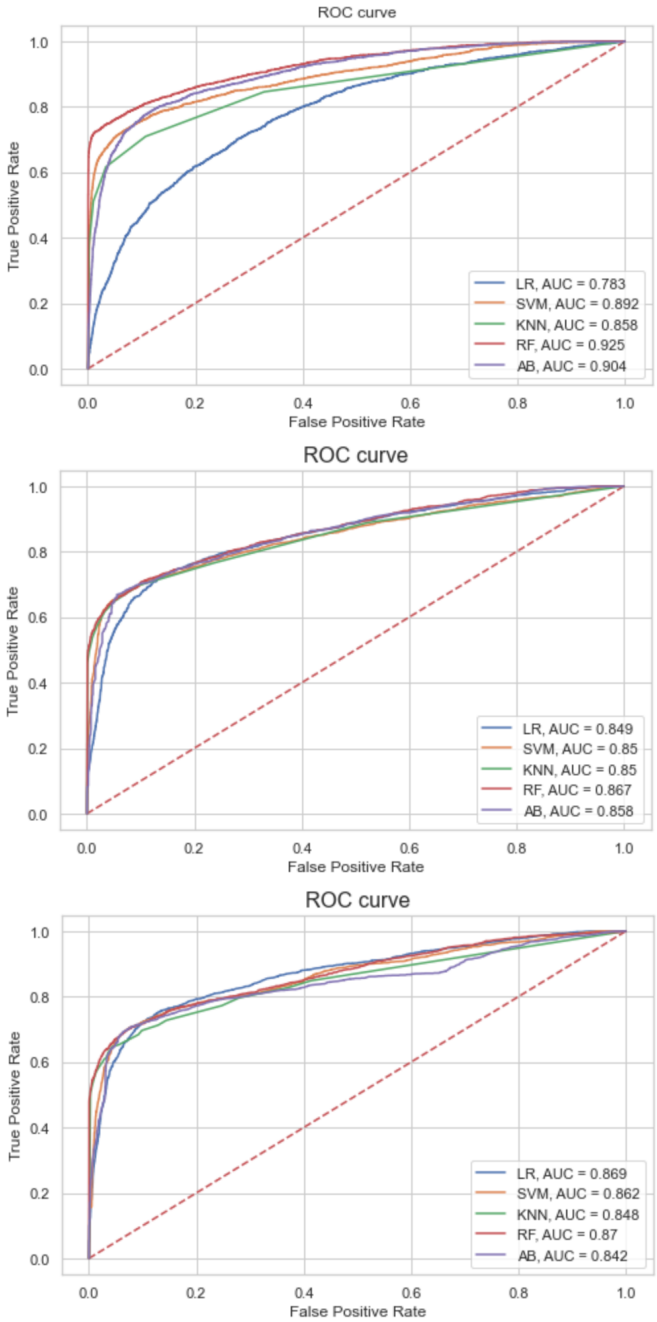

Figure 4.2. From top to bottom: ROC curves for individual models based on entire data set, ROC curves for individual models based on features selected data set and ROC curves for cluster-based models on features selected data set.

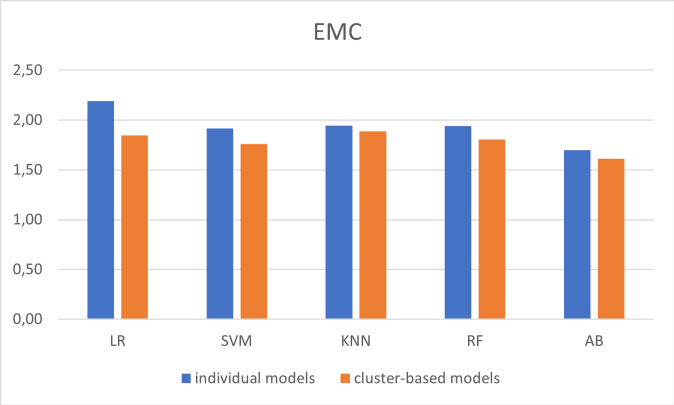

Figure 4.3. Expected misclassification error both for individual models (in blue) and for cluster-based models (in orange), applied on feature selected data set.

Chapter 5

Conclusion

The objective of this work was to develop and compare different machine learning models in order to classify clients as default or non-default ones, and then to incorporate unsupervised learning techniques at clustering stage to try to improve the previous performances.
According to the results reported in the previous chapter, we can say that there is no one best individual model between the five tested. The goodness of each method, in fact, depends on the performance measure we want to optimize and on the number of features used to predict the response. However, we can conclude that the usage of unsupervised learning deeply contribute to the improvement of models results. This improvements can be observed not only from the point of view of the seven different performance measures considered but also from an economical point of view (i.e. expected misclassification cost).
It is important to observe that in our work we only used feature selection to select fewer feautures. It might be interesting to try using dimensionality reduction methods (such as PCA) instead. Of course we will lose in terms of interpretability but performance improvements could be observed.
Furthermore, potential future works could include the study of a consensus model and the application of unsupervised learning techniques also in the

consensus stage. For example, if would be interesting to develop a consensus classification decision based on predictive outcomes of individual ML models. Several consensus techniques could be considered, one of the most commonly used and most simple is the majority vote (MV). At this point it would be possible to add a new unsupervised learner such as Kohonen's self-organizing map (SOM), which is an unsupervised neural network introduced by Kohonen. Its idea is to project a nonlinear data vector from an high-dimensional space to a bi-dimensional one.

Another improvement can be made in terms of maximizing or minimizing some measure of performance such as EMC. In fact, to compare different models, in order to switch from the probability of default to a binary output, we decided to set a discriminating threshold equal to 0.5 for all methods. However, if we consider a single model, this is not the best we can do. Once selected a good model,it would be nice to modify his threshold such as the EMC or any other measure of our interest would be optimized. If, for example, we would like to minimize the economic impact (in terms of EMC) of our model, it is possible to construct a neural network that, once we fixed C_p and C_n, selects the best threshold possible.

Bibliography

[1] Alexander J. McNeil, Rudiger Frey, Paul Embrechts, Quantitative RIsk Management: Concept, Techiniques and Tools, Princeton University Press, 2005

[2] Camilla Calì, Maria Longobardi, Some mathematical properties of the ROC curve and their applications, Springer, 2015.

[3] Capon, Noel, Credit Scoring Systems: A Critical Analysis, Journal of Marketing, Spring, 1982.

[4] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, An Introduction to Statistical Learning, Springer, 2013.

[5] Lyn C. Thomas, David B. Edelman, Jonathan N. Crook, Readings in Credit Scoring - Foundations, developments, and aims, Oxford University Press, 2004.

[6] Lyn C. Thomas, David B. Edelman, Jonathan N. Crook, Credit Scoring and Its Applications, SIAM, 2002.

[7] Marc Peter Deisenroth, A. Aldo Faisal, Cheng Soon Ong, Mathematics for Machine Learning, Cambridge University Press, 2020.

[8] Mehryar Mohri, Afshin Rostamizadeh, Ameet Talwalkar, Foundations of Machine Learning, MIT Press, 2012.

[9] Ning Chen, Bernardete Ribeiro, An Chen, Financial credit risk assessment: a recent review, Springer Science+Business Media Dordrecht, 2015.

[10] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, W. Philip Kegelmeyer, SMOTE: Synthetic Minority Over-sampling Technique, Journal of Artificial Intelligence Research, 2002.

[11] Oded Maimon, Lior Rokach, Data Mining and Knowledge Discovery Handbook, Springer, 2010.

[12] Oesterreichische Nationalbank (OeNB), Rating Models and Validation - Guidelines on Credit Risk Management, 2004.

[13] Panayiota Koulafetis, Modern Credit Risk Management - Theory and Practice, Palgrave Macmillan, 2017.

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

[15] Wang Bao, Ning Lianju, Kong Yue, Integration of unsupervised and supervised machine learning algorithms for credit risk assessment, 2019, Expert Systems With Applications, 128(2019), 301-315.