Highly imbalanced learning, application to fraud detection

Rémi Viola

École Doctorale ED488 Sciences, Ingénierie, Santé
Numéro d'ordre NNT : 2022LYSES021

Highly Imbalanced Learning Application to Fraud Detection

Apprentissage Fortement Déséquilibré Application à la Détection de Fraudes

Thèse préparée par **Rémi VIOLA**
au sein de l'**Université Jean Monnet de Saint-Étienne**
en collaboration avec la **Direction Générale des Finances Publiques**
pour obtenir le grade de :

Docteur de l'Université de Lyon
Spécialité : **Informatique**

Université de Lyon, UJM-Saint-Etienne, CNRS, Institut d'Optique Graduate School,
Laboratoire Hubert Curien UMR 5516, F-42023, SAINT-ETIENNE, FRANCE.

Direction Générale des Finances Publiques, Bureau SJCF-1D,
Bâtiment Turgot, 86-92 allée de Bercy, 75574, PARIS, FRANCE.

Thèse soutenue publiquement le 24/06/2022 devant le jury composé de :

Élisa FROMONT	Professeure, Université de Rennes	Rapporteuse
Amaury HABRARD	Professeur, Université de Saint-Étienne	Co-Directeur
Jean-Christophe JANODET	Professeur, Université d'Évry	Rapporteur
Marc SEBBAN	Professeur, Université de Saint-Étienne	Directeur
Christine SOLNON	Professeure, INSA de Lyon	Présidente
Haïcheng TAO	Responsable Data Science, DGFiP Paris	Co-Encadrant

Remerciements

Que de chemin parcouru depuis 2013 et cette formation Éducation Nationale pour enseigner l'informatique en Lycée. Finalement, heureusement qu'elle était trop superficielle car cela m'a permis de me retrouver à écrire ces remerciements aujourd'hui. Histoire de rendre à chacun les mérites qu'il a dans cette grande aventure, je vais donc essayer de les remercier par ordre chronologique. Cela me permet en même temps de vous raconter cette histoire.

Je vais donc commencer par remercier Hugo Lefeuvre, alors élève en 1S au lycée Honoré d'Urfé à Saint-Étienne et, aux dernières nouvelles, doctorant dans l'équipe Advanced Processor Technologies à l'université de Manchester. Sans lui, je ne me serais jamais retrouver ici... En ayant un niveau largement supérieur au mien à la sortie de ma formation, il m'a poussé à me remettre en question sur ma capacité à lui enseigner quelque chose et ainsi poussé à m'inscrire en Licence bien malgré lui.

Après avoir donc fait une L2 Informatique en autonomie, j'ai surtout fait une L3 pour me mettre au niveau. C'est là que j'ai eu le plaisir et la chance de rencontrer des professeurs inspirants parmi lesquels je citerai surtout Marc Sebban, qui m'a suivi tout au long de cette période, et Elisa Fromont. Leurs cours étaient intéressants et bien menés. C'était un réel plaisir de les retrouver chaque semaine. Et, alors que j'avais seulement envisagé de faire une année de mise à niveau pour le lycée, ils ont réussi à me vendre leur Master Machine Learning and Data Mining, lors de leur présentation de la filière. C'est comme ça que je me suis retrouvé à m'inscrire en MLDM et je les en remercie. Je remercierai par la même occasion l'ensemble des enseignants du Master et de l'équipe Data Intelligence du Laboratoire Hubert Curien, dont Amaury Habrard qui, avec Marc, à accepter d'encadrer cette thèse.

Lors de ces trois années de L3 et Master, j'ai également eu la chance de travailler avec de nombreux étudiants lors de différents projets et c'est aussi en partie grâce à eux que tout ceci a été possible. Je ne peux bien évidement pas tous les remercier ici, je me contenterai donc des deux principaux. En premier lieu, je souhaite remercier Valentin, l'un de mes plus fidèles compagnons de projets. Il m'a beaucoup apporté et je suis content d'avoir pu le rencontrer. Mais le plus important reste mon petit frère Mathieu Viola. Ensemble, nous avons réussi cette reprise d'étude commune, afin de donner une nouvelle impulsion à notre vie professionnelle. Sans lui, rien de tout ceci n'aurait été possible... Il m'était difficile de gérer l'autonomie en L2 en parallèle du travail, alors il me fournissait les cours que je travaillais le soir et le week-end. Il m'a également énormément aidé pour les cours de L3 ou de Master que je manquais à cause de mon mi-temps au lycée. En filmant notamment ceux de Master, il m'a permis de ne pas perdre le fil. Je suis content d'avoir pu partager cela avec lui et je ne l'en remercierai jamais assez.

Ensuite, je souhaite remercier la DGFiP et les différents représentants du bureau SJCF-1D qui ont proposé cette thèse à Marc et au Laboratoire Hubert Curien. Mon aventure avec eux a réellement commencé en 2018 lors de mon stage de M2 qui a servi de tremplin à cette thèse. Ainsi, Sébastien Rioux et Philippe Schall m'ont fait confiance pour améliorer un de leurs modèles, ce qui m'a permis de mettre un premier pied dans la Mission Requêtes et Valorisation(MRV). Sébastien avait alors pris le temps de m'expliquer le fonctionnement de la MRV et du contrôle fiscal en général, ce qui a suscité en moi de plus en plus d'intérêt pour ce travail. Par la suite, en acceptant mon détachement au Ministère de l'Économie et des Finances, ils

m'ont permis de gérer la thèse plus sereinement. J'en profite aussi pour remercier tous les autres membres de la MRV qui ont travaillé avec moi lors de ces 4 années, notamment Agathe Muselet avec qui j'ai beaucoup échangé sur ses modèles, Victor Ng qui a repris ma première contribution dans un de ces modèles, Cécile Lefevre-Ardant avec qui j'avais plaisir à échanger sur les différents sujets qui nous liaient et qui m'a en partie transmis le projet Réseaux Sociaux à son départ ou encore Haicheng Tao qui dirige maintenant l'équipe à la place de Sébastien. Pour n'oublier personne de l'équipe je remercierai également Alexia Suchet et Jean Cabouat avec qui je n'ai pas encore vraiment eu la chance de travailler mais cela ne saurait tarder. Avec le prolongement de mon détachement pour rester avec eux, cela devrait sûrement arriver.

En parallèle de l'équipe MRV, il y a l'ensemble des doctorants et personnels que j'ai eu la chance de rencontrer au Laboratoire Hubert Curien. Ils m'ont épaulé, écouté et permis d'avancer et je n'en serai sûrement pas là aujourd'hui s'ils n'avaient pas été là. Je remercierai en premier lieu Jordan qui m'a inspiré lors de mon stage de M2. Ses explications sur le Gradient Boosting et l'Average Precision ont été des plus précieuses pour le travail que j'avais à mener à ce moment là. Valentina a, quant à elle, su maintenir l'ordre dans ce bureau rempli de garçons pas toujours très concentrés, ce qui nous a permis de maintenir le cap. Thomas qui était régulièrement mon support technique quand j'avais des problèmes sur les machines de calculs, m'a également beaucoup apporté. Il faut également citer Paul, Rémi, Jules et Raphaël, qui, même si je les ai un peu moins côtoyés, font partie intégrante de cette histoire et j'ai eu plaisir à discuter avec eux. Cependant, quatre doctorants ont une place plus importante sur cette période. Tanguy m'a appris beaucoup de choses sur le transport optimal et les discussions qu'on avait étaient souvent très enrichissantes. Ensemble, nous avons encadré un stagiaire, Thibaud, qui a finalement aussi pris part à cette aventure. J'ai trouvé cette période très intéressante car, même si cela m'a retardé dans ma rédaction de la thèse, cela m'a fait avancer du côté de mon travail à la DGFiP. Kévin a toujours été présent pour discuter et se changer les idées. Les seuls regrets que j'ai avec lui, c'est, d'une part, que l'on n'ait pas pu travailler directement ensemble et d'autre part, qu'il ne soit jamais venu en soirée avec nous. Enfin, mention spéciale à Léo et Guillaume, avec qui j'ai eu la chance de discuter, d'échanger, de travailler et de rédiger des articles. Chacun d'eux m'a apporté ce qu'il pouvait et m'a fait avancer, l'un par sa capacité à coder de manière optimale, en résolvant tous les petits problème que je rencontrais et l'autre en m'apportant son expertise théorique sur les sujets que je ne maîtrisais pas ou plus. Sans eux, je ne serai pas en mesure de soutenir cette thèse et je les en remercie du fond du cœur.

A ce stade, je tiens maintenant à remercier les membres de mon jury. D'abord je remercierai une nouvelle fois Marc et Amaury d'avoir accepté d'encadrer cette thèse. Merci pour l'opportunité que vous m'avez offert, elle a définitivement changé ma vie. Merci aussi à Elisa Fromont et Jean-Christophe Janodet d'avoir bien voulu être les rapporteurs de cette thèse. Enfin, je remercie Christine Solnon d'être la présidente de ce jury.

Finalement, je remercierai la personne la plus importante à mes yeux, Perrine... Sans elle, tout aurait été beaucoup plus compliqué. En acceptant ma reprise d'étude, elle ne se doutait pas de ce qui allait arriver. Elle a été mon soutien indéfectible et m'a supporté pendant tout ce temps, et cela n'a pas été tous les jours facile... Elle a tout géré pendant que je n'étais pas là, notamment les enfants, et je l'a remercie pour tous les sacrifices auxquels elle a consenti pour que je puisse y arriver. Maintenant que cette thèse est terminée, j'espère pouvoir lui renvoyer l'ascenseur. Dans tous les cas, tout ce que je pourrais écrire ne suffirait jamais à la remercier suffisamment. Je me contenterai donc de finir par ces quelques mots : Merci infiniment pour ta présence et ton soutien, je t'aime...

Contents

Introduction

In the domain of Artificial Intelligence, Machine Learning encompasses methods that allow a system to learn from training data and address various kinds of problems. Learning techniques often differ by the nature of the collected data which can be numerical, categorical, structured and take a large diversity of forms, from images to videos, texts, signals, graphs, time series, etc. The task to be accomplished on these data is a key ingredient of the Machine Learning process. One might want to group instances in an unsupervised fashion according to some (dis)similarity measure, thus addressing a clustering task. Or one might be interested, like in this thesis, in predicting in a supervised way a specific label assigned to each instance. This label can be either discrete as encountered in a classification task or continuous as in a regression scenario. In the former case, when only two possible labels are involved, like in fraud detection, the topic at the core of this thesis, we usually use the value +1 for describing fraudulent transactions and -1 for genuine ones. The presence of such a supervision in the form of labeled training examples is of great interest for guiding the optimization process of the model parameters. In this thesis, we will focus on supervised binary classification tasks in the specific context of tax returns fraud detection.

Fraud detection is a major problem for most governments. Indeed, the huge financial losses penalize them in the execution of their public policies, in Education, Justice or Health. In France, the 2019 report of the "Cour des Comptes" [31] estimates the amount of VAT fraud at around 15 billion euros, which is more than the budget allocated to the Ministry of Solidarity and Health this year... This report also estimates that the amount of Social Security fraud is more than 9 billion euros, which is slightly less than the budget allocated to the Ministry of Justice... This shows why fraud detection has become a key challenge nowadays, not only for governments but also for banks or insurance companies. Thus, for some time now, there has been a growing interest in research on this topic, especially in Machine Learning [2].

However, fraud detection is a particularly challenging task in general, mainly because the examples of interest, *i.e.* the frauds also called positive examples, are very often much less numerous than the negative examples, corresponding to the non fraudsters. For this reason, fraud detection is an application that falls into the scope of the so-called Imbalanced Machine Learning setting. From a scientific perspective, the scarcity of the positive examples makes the problem interesting, opening the door to new methodological contributions from the Machine Learning community.

Imbalanced Learning also encompasses anomaly detection problems which covers a large spectrum of application such as in medicine, computer networks or weather forecasts. However, while anomaly detection and fraud detection seem similar at first glance, they differ in many aspects which make the techniques associated to the two domains very different. Indeed, anomalies are somewhat 'just' some kind of unexpected events with respect to some trend. On the other hand, fraud detection can be a much more challenging task since for example fraudsters can try to mimic the best genuine behavior to remain undetected. In this thesis, we propose to address this challenging problem by developping new contributions that are evaluated in the context of a specific application representing the main thread of this document and detailed in

the next paragraphs.

Context of this thesis. This thesis was carried out at the General Directorate of Public Finances (DGFiP) which is part of the French Ministry of Economy and Finance and in the Data Intelligence team at the Hubert Curien Laboratory which is a joint research unit (UMR 5516) between the University of Saint-Etienne, the University of Lyon, the CNRS and the Institut d'Optique Graduate School. It was funded by the DGFiP.

About the DGFiP The Controls Programmation and Data Analysis section of the SJCF-1D office is part of the DGFiP and is based in Paris. It is mainly in charge of fraud detection and query management projects. The objective is to improve the efficiency of tax control operations by renovating the targeting phase of the operations upstream of the control process. This step largely determines the results of the entire tax control chain. The DGFiP has a large volume of data in various fields, concerning both companies and individuals. This is why it has been modernizing its analysis techniques for several years. Since 2017, they use Data Mining and Machine Learning for analyzing and cross-referencing all the information available to the DGFiP in order to identify, through statistical or mathematical methods, the criteria that characterize a fraudulent person or company and thus establish a fraud profile that will be applied to a target population. For companies, for example, datasets can gather data from income, tax and VAT returns and are used to detect different types of tax fraud such as overestimated or fake charge returns, revenue minimization or international VAT frauds such as "VAT carousels" [1]. The DGFiP performs about 50,000 tax audits per year within a panel covering more than 3,000,000 companies. Being able to select the right companies to control each quarter is a crucial issue with a potential high societal impact. Since 2018, the percentage of controls from the lists provided by the SJCF-1D office has been steadily increasing and is expected to reach at least half of these controls by 2022. In 2019, 22% of tax control operations were scheduled by the office and they resulted in the recovery of a total of 785 million euros. In 2021, the proportion of audits scheduled by the office should reach 42%.

The overall goal of the DGFiP is to use artificial intelligence to better target the companies to be controlled, to detect more complex and sophisticated cases of fraud, which would be difficult or even impossible to detect manually by human beings, in order to diversify the angles of attack against fraudsters.

The objective of this thesis is to design new Machine Learning models and algorithms for addressing this challenging task of highly imbalanced learning. Taking into account the constraints imposed by the applied context of fraud detection, our contributions have been developped according to three main criteria:

1. **Efficiency**: All along this thesis, we kept in mind that the capacity of our models to identify fraudsters will be evaluated eventually in terms of the amount of money recovered by the DGFiP.

2. **Budget constraints**: The number of possible tax audits per year remains small compared to the huge number of companies. The optimization of specific criteria allowing us to take into account the limited number of controllers has guided our contributions.

3. **Interpretability**: In fraud detection, models based on Machine Learning mainly aim at supporting decision making. Being able to explain the alerts generated (in the form of ranking) by the automatic system becomes more and more important at the DGFiP.

[1] `https://en.wikipedia.org/wiki/Missing_trader_fraud`

Outline of the thesis. These objectives in mind, this manuscript presents, after a first chapter reviewing the notions and methods in Machine Learning required for the understanding of the rest of the document, three main contributions.

In Chapter 2, we address the problem of highly imbalanced learning from a geometric perspective. We propose a simple and intuitive modification of the popular *k-Nearest Neighbor* algorithm to modify the decision boundaries in favor of the minority class. By reweighting the distances to positive examples using a parameter γ, our method allows one to reduce the False Negative rate (*i.e.* frauds that would be missed) while controlling the risk to get False Positives (*i.e.* non fraudulent companies that would be controlled). From a geometric point of view, reweighting by γ the distances to positive examples boils down to applying a constant distortion of the feature space leading to an extension of the zone of influence of the minority class. If we take a step back, we can see that this process is in fact a special case of Mahalanobis Metric Learning which aims at optimizing a linear transformation leading to ellipsoids.

In Chapter 3, we generalize this idea by designing a new Metric Learning algorithm dedicated to improve a *k-NN* classifier in highly imbalanced scenarios. By using the Uniform Stability framework, the resulting method comes with theoretical guarantees on the False Positive and False Negative rates.

In Chapter 4, our last contribution directly addresses the increasing need of the DGFiP to have access to interpretable ranking models. We present *MetaAP*, a meta-tree-based ranking algorithm which directly optimizes the *Average Precision*. This latter has been shown to be very efficient to move positive examples at the very top of the ranking, the only part of the list that will be really exploited by the limited number of controllers. Our model also comes with the nice feature of generating compact rules supporting the decision making of the agents of the DGFiP.

The contributions presented in this thesis led to the following publications:

Publications in International Conferences

Rémi Viola, Rémi Emonet, Amaury Habrard, Guillaume Metzler and Marc Sebban. Learning from Few Positive: a Provably Accurate Metric Learning Algorithm to Deal with Imbalanced Data. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2020, Virtual, Japan [118].

Rémi Viola, Rémi Emonet, Amaury Habrard, Guillaume Metzler, Sébastien Riou and Marc Sebban. An Adjusted Nearest Neighbor Algorithm Maximizing the F-Measure from Imbalanced Data. In *IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 2019, Portland, United States [113].

Publications in Journals

Rémi Viola, Léo Gautheron, Amaury Habrard and Marc Sebban. MetaAP: a Meta-Tree-Based Ranking Algorithm Optimizing the Average Precision From Imbalanced Data. In *Pattern Recognition Letters*, (revised version under review). 2022 [116].

Rémi Viola, Rémi Emonet, Amaury Habrard, Guillaume Metzler, Sébastien Riou and Marc Sebban. A Nearest Neighbor Algorithm for Imbalanced Classification. In *International Journal on Artificial Intelligence Tools*, volume 30, doi:10.1142/S0218213021500135. 2021 [117].

Communications in National Conferences

Rémi Viola, Rémi Emonet, Amaury Habrard, Guillaume Metzler and Marc Sebban. MLFP: Un Algorithme d'apprentissage de Métrique pour la Classification de données déséquilibrées.

In *Conférence sur l'Apprentissage automatique (CAp)*, 2020, Virtual, France [115].

Rémi Viola, Rémi Emonet, Amaury Habrard, Guillaume Metzler, Sébastien Riou and Marc Sebban. Une Version Corrigée de l'Algorithme des Plus Proches Voisins pour l'Optimisation de la F-Mesure dans un Contexte Déséquilibré. In *Conférence sur l'Apprentissage automatique (CAp)*, 2019, Toulouse, France [114].

List of Notations

Notations	Descriptions
S	a Sample
m	Number of examples
$\mathbf{x}$	a vector
x_i	the i-th element of the vector $\mathbf{x}$
$\mathcal{X}$	The input space
$\mathcal{Y}$	The ouput space
$\mathcal{Z}$	The joint space with $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$
d	Dimension of the feature space
$\mathcal{D}$	a distance
k	an integer
$\mathbf{M}$	a matrix
$\mathbb{S}^+$	the set of Positive Semi Definite matrix

Chapter 1

Basics of Machine Learning

Abstract

In this chapter, we present the basics of Supervised Machine Learning, from the notions of loss function and risk minimization to the theoretical guarantees that can be derived from finite training sets. We also present some of the most well-known Machine Learning algorithms including the Nearest-Neighbors and Decision Trees that are at the heart of our main contributions. Moreover, we introduce the Metric Learning setting which also plays an important role in this thesis. Finally, we detail the specificities of Imbalanced Learning which caracterizes the problem of fraud detection.

1.1 Supervised Machine Learning

1.1.1 Generalities on Machine Learning

Like human learning, Machine Learning is often example-based. By exploiting collected training data, which are the key ingredient of the entire process, Machine Learning gives us different ways to infer rules and models to avoid solving some tasks by hand. Machine Learning can be basically (and non exhaustively) divided into three main fields: Supervised Learning, Unsupervised Learning, and Reinforcement Learning [85].

In Supervised Learning, the manipulated data are either annotated with a discrete or continuous label. The nature of these labels allows us to solve different kinds of tasks including classification, regression or ranking to cite a few of them. All these settings share a common feature: they aim at optimizing the parameters of a so-called hypothesis which fits the best the training data, according to some loss function. Once learned, the induced model can be deployed on new unknown data. Among the most popular algorithms, we can mention the $k-$Nearest Neighbors ($k-$NN) [42], Decision Trees [72, 90, 91], Support Vector Machines (SVM) [12, 28, 112], Logistic Regression [11], Boosting [98, 45] or Neural Networks [82, 93, 95].

In Unsupervised Learning, we cannot benefit from labels. The idea is then to discover the underlying structure of the data. For instance, we can use the similarities in the examples (i) to group them together and perform a clustering task, (ii) to approximate the underlying distribution of the data by density estimation or (iii) to project the data from a high-dimensional to a smaller space, by dimensionality reduction. Among the popular algorithms, we can name $k-$Means [77], Principal Component Analysis (PCA) [87, 60] or Matrix Factorization [70].

Finally, in Reinforcement Learning [105], the idea is to find the action or sequence of actions to perform in a particular context, called a policy, in order to maximise a certain reward. In this case, the model does not have examples to learn from but proceeds on the principle of trials and errors to determine the optimal strategy, like a child who learns to stand upright but who falls a lot before reaching his goal. Popular algorithms include Q-Learning [122] and its Deep extensions [84], SARSA [96] or the Bandits approaches [18].

In this thesis we will only focus on Supervised Learning for binary classification and ranking tasks.

1.1.2 Data as a key ingredient of Machine Learning

As mentioned before, data is essential in the whole Machine Learning process. In particular, two important points deserve to be emphasized about the number of required training examples.

First, as formally described later in the generalization bounds, the number of data is key in Supervised Learning for ensuring a capacity for the hypothesis to correctly classify new data at test time. Indeed, without enough data, the algorithm could learn a classifier that would not reflect the reality of the distribution, as can be seen in Figure 1.1, where the decision boundaries are supposed to be a square centered at zero. On this task, a SVM (see Section 1.3.2) fails to find the correct hypothesis when the number of examples is small (left). It behaves better as this number grows (center) and is almost perfect when the size of the training data set is large (right).

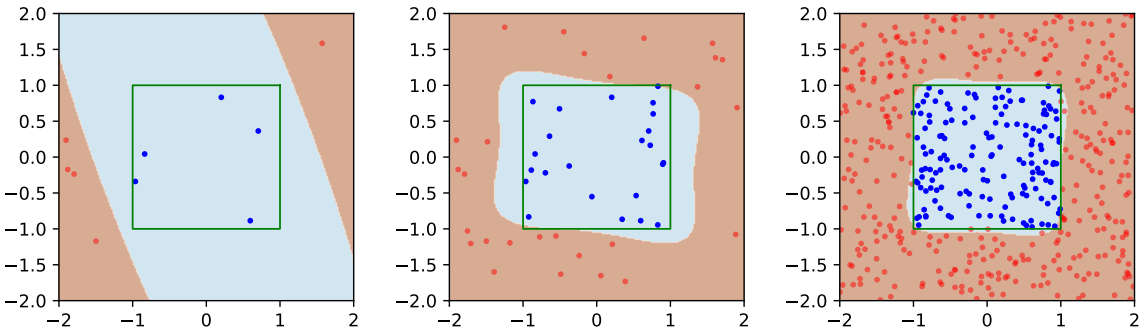

Figure 1.1: Impact of the number of training examples in the learning process. In each figure, the blue points correspond to positive examples and the red points to negatives. Positives are uniformly draw from the target green square. The decision boundaries of the learned SVM define blue and red regions correponding to a prediction $+1$ and -1 respectively. In each of the three cases, the hypothesis is perfect on the training data but behaves poorly at test time for the first two. On the right figure, the number of training examples is sufficient to retrieve the underlying distribution and then classify correctly at test time.

The second important point is related to the notion of 'Curse of Dimensionality'. It corresponds to the fact that as the number of features describing our problem increases, the number of training examples has to grow exponentially fast. As shown in Figure 1.2, if we want to cover the $[0, 1]$ interval (1D) uniformly, we need 11 points to have a distance of 0.1 between each point. In a two dimensional space, the number of points needed is $11^2 = 121$. In 3 dimensions, $1,331$ points are required and in 10 dimensions, almost 26 billion!

Note that if the number of training examples is not large enough, the learning algorithm might face a so-called overfitting phenomenon consisting in selecting an excessively complex model, with possibly too many parameters, that would lead to a poor behavior at test time.

1.1.3 Loss functions and Risk Minimization

We consider a set $S = \{\mathbf{z}_i = (\mathbf{x}_i, y_i)\}_{i=1}^m$ of m training samples, independently drawn and identically distributed from an unknown joint distribution $\mathcal{D}_\mathcal{Z}$, over a space $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X} = \mathbb{R}^d$ is the feature space and $\mathcal{Y} = \{-1, +1\}$ is the set of labels. We assume that $S = S_+ \cup S_-$ with m_+ positives in S_+, m_- negatives in S_- and $m = m_+ + m_-$.

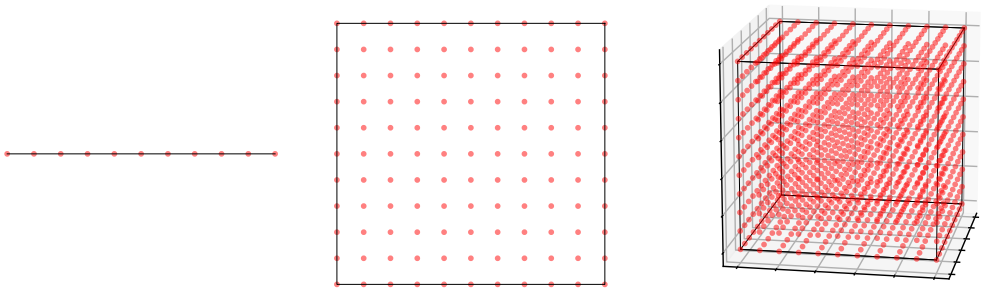

Figure 1.2: Impact of the dimension on the needed number of training examples. In each figure, the black lines delimit the area to be covered uniformly with the red points. On the left figure, 11 points are needed to cover the segment uniformly. In the middle, 121 points are needed for the square. On the right figure, 1,331 points are used to cover the cube.

We also assume that there exists a target function f such that $y = f(\mathbf{x})$, $\forall(\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}$. If we call $\mathcal{H}$, the set of hypotheses, *i.e.* a set of possible functions to approximate f, the goal of a Supervised Learning algorithm is to output from S the best hypothesis as close to the target function f as possible.

Loss functions To pick the best candidate, we need a criterion to evaluate the quality of each tested hypothesis $h \in \mathcal{H} : \mathcal{X} \to \mathbb{R}$. For this purpose, we consider a nonnegative loss function $l : \mathcal{H} \times \mathcal{Z} \to \mathbb{R}^+$ measuring the degree of disagreement between $h(\mathbf{x})$ and y. Among the possible loss functions, the most natural one is the $0/1$ loss, represented in Figure 1.3 that counts the number of classification errors made by the hypothesis h:

$$l_{0/1}(h, z) = \begin{cases} 1 & \text{if } y \times h(\mathbf{x}) < 0 \\ 0 & \text{otherwise.} \end{cases}$$

Due to the fact that the $0/1$ loss is nonconvex, surrogate convex loss functions are often used in practice. The most popular ones are:

- the Hinge loss used in Support Vector Machine: $l_{hinge}(h, \mathbf{z}) = \max(0, 1 - y \times h(\mathbf{x}))$,
- the Exponential loss used in Boosting: $l_{exp}(h, \mathbf{z}) = \exp^{-y \times h(\mathbf{x})}$,
- or the Logistic loss used in Logistic Regression: $l_{log}(h, \mathbf{z}) = \log\left(1 + \exp^{-y \times h(\mathbf{x})}\right)$.

True Risk and Empirical Risk Given this nonnegative loss function $l : \mathcal{H} \times \mathcal{Z} \to \mathbb{R}^+$, we can define the **True Risk** of h with respect to l, as the expected loss suffered by h over the distribution $\mathcal{D}_{\mathcal{Z}}$:

$$\mathcal{R}^l(h) = \mathbb{E}_{z \sim \mathcal{D}_{\mathcal{Z}}}\left[l(h, z)\right]$$

The optimal hypothesis h^* can be defined as follows:

$$h^* = \arg\min_{h \in \mathcal{H}} \mathcal{R}^l(h)$$

Since the computation of the True Risk is not possible because $\mathcal{D}_{\mathcal{Z}}$ is unknown, we cannot directly select h^*. One way to overcome this issue consists in using the empirical counterpart of $\mathcal{R}^l(h)$, called the Empirical Risk and computed from the training set S:

$$\widehat{\mathcal{R}}^l(h) = \frac{1}{m} \sum_{i=1}^{m} \left[l(h, z_i)\right]$$

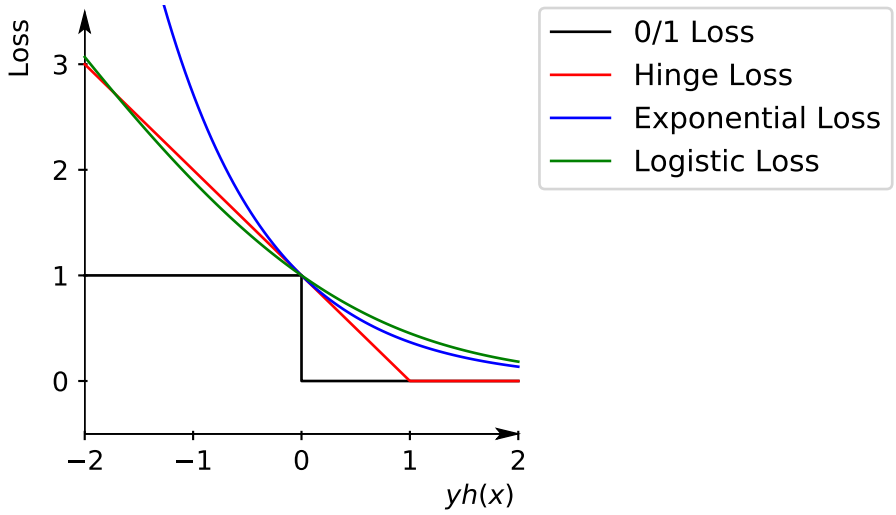

Figure 1.3: Different loss functions. In black, the natural 0/1 loss. In red, the Hinge Loss used in the SVM algorithm. In blue, the Exponential loss used in Boosting. In green, the Logistic loss used in Logistic Regression.

A common strategy is then to find the hypothesis h that minimizes this Empirical Risk. This strategy is called **Empirical Risk Minimization** and is defined as follows:

$$h = \arg\min_{h_i \in \mathcal{H}} \widehat{\mathcal{R}}^l(h_i) \tag{1.1}$$

Occam's Razor Principle It is important to note that several hypotheses can be consistent with S. Thus, for two hypotheses with similar performances, Occam's Razor principle suggests to choose the simplest one. Following this principle tends to avoid overfitting phenomena.

One way to do it in practice is to add a regularization term to the Problem 1.1 to penalize the complexity of the hypothesis. A Supervised Machine Learning problem can then typically be defined as follows:

$$h = \arg\min_{h_\theta \in \mathcal{H}} \widehat{\mathcal{R}}^l(h_\theta) + \lambda \|\theta\|_p^p \tag{1.2}$$

where λ is a regularization parameter and $\|.\|_p$ is a p-norm over the parameters θ of the hypothesis h_θ. The selected hypothesis h is the one offering the best trade-off between empirical risk minimization and simplicity. This strategy, called **Regularized Risk Minimization**, limits the risk of overfitting and can also help to solve ill-posed problems when using strongly convex regularizations and thus speed-up the calculation of the solution.

Bias/Variance trade-off There are two major sources of possible errors between the selected hypothesis h and the target function f.

The first one is the **Inductive Bias**. Indeed, there is no guarantee that the chosen hypothesis space $\mathcal{H}$ will match the target function f, even if we would be able to find the optimal hypothesis $h^* \in \mathcal{H}$. The bias expresses the deviation between h^* and f.

The second source of error is the **Variance**. As the model is learned on a finite randomly drawn data set, the optimal hypothesis h^* is usually not selected.

Figure 1.4 illustrates the trade-off between the bias and the variance. As the complexity of the model increases, the expressiveness of the hypothesis is improved leading to a reduction of the bias. But this happens at the expense of an increase of the variance due to the expansion of the set of parameters to be optimized. Learning well boils down to finding a good compromise between a too simple model (large bias, small variance) that will tend to underfit the data, and a too expressive model (small bias, large variance) which will suffer from an overfitting phenomenon.

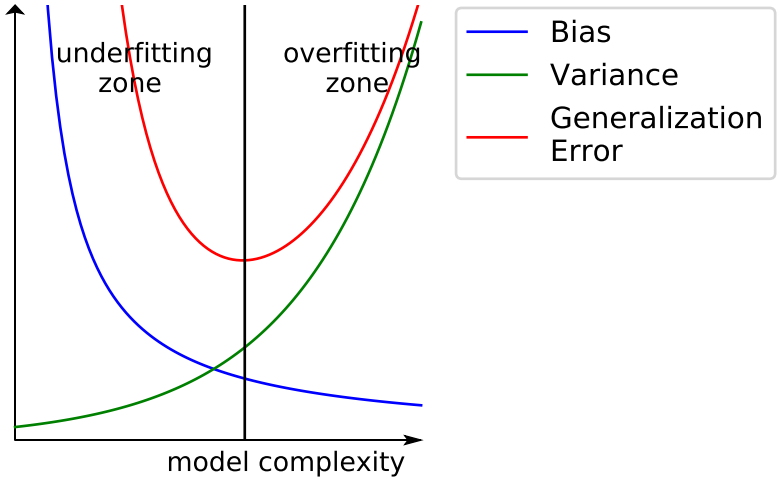

Figure 1.4: Illustration of the Bias/Variance trade-off. With a too simple model, we will have a low variance but a high bias. It corresponds to a situation of underfitting. With a too complex model, we will have a low bias but a high variance. It corresponds to a situation of overfitting. A trade-off has to be found between the bias and the variance to have a better model.

1.1.4 Generalization Guarantees

We saw earlier that the learning process is done by minimizing the **Empirical Risk** on the training data S, under the assumption that this set is sufficiently large and representative of the underlying unknown distribution $\mathcal{D}_{\mathcal{Z}}$. During the last decades, many theoretical frameworks have been developed to derive generalization guarantees on the learned hypothesis h [111, 110, 13, 127].

Based on the work of Leslie Valiant [110], the PAC theory, for Probably Approximatively Correct, gives us the general form of a generalization bound which bounds the probability to observe a large deviation between the True Risk and the Empirical Risk of h.

$$P\left(\left|\mathcal{R}^l(h) - \widehat{\mathcal{R}}^l(h)\right| \geq \epsilon\right) \leq \delta$$

where $\epsilon > 0$ and $\delta \in [0, 1]$.

Different frameworks have been proposed in the literature, most of them resorting to concentration inequalities to derive specific bounds. In the following, we enter into the details of the Uniform Stability framework that has been used to prove guarantees on our Metric Learning algorithm presented in Chapter 3.

Uniform Stability The Uniform Stability framework [13] has the particularity to provide generalization bounds that are specific to the learning algorithm, *i.e.* that take into account in particular the nature of the loss function, the model considered and the regularization term.

This framework tries to find out under which conditions an algorithm is uniformly stable. An algorithm is said to be stable when its predictions do not change significantly in case of minor modifications of the training set.

Definition 1. *An algorithm $\mathcal{A}$ has a uniform stability in $\frac{\kappa}{m}$ with respect to a loss function l and a parameter set θ, with κ a positive constant if:*

$$\forall S, \forall i, 1 \leq i \leq m, \sup_{\mathbf{z}} |l(\theta_S, \mathbf{z}) - l(\theta_{S^i}, \mathbf{z})| \leq \frac{\kappa}{m}$$

with S a learning sample of size m, θ_S the model parameters learned from S, θ_{S^i} the model parameters learned from the sample S^i obtained by replacing the i^{th} example $\mathbf{z}_i$ from S by another example $\mathbf{z}'_i$ independent from S and drawn from $\mathcal{D}_\mathcal{Z}$. Finally, $l(\theta_S, \mathbf{z})$ is the loss suffered at $\mathbf{z}$.

In [13], the authors provide a generalization bound when an algorithm fulfills the uniform stability criterion:

Theorem 1. *Let $\mathcal{A}$ be an algorithm with uniform stability in $\frac{\kappa}{m}$ with respect to a loss function l bounded by K. Then, for any $m \geq 1$, and any $\delta \in [0, 1]$, the following bound holds with probability at least $1 - \delta$ over the random draw of the sample S,*

$$\mathcal{R}^l(\theta_S) \leq \widehat{\mathcal{R}}^l(\theta_S) + \frac{2\kappa}{m} + (4\kappa + K)\sqrt{\frac{\ln 1/\delta}{2m}}.$$

Note that this bound is in $\mathcal{O}\left(\frac{1}{\sqrt{m}}\right)$ so, as expected, the greater the number of examples, the smaller the difference between the **Empirical Risk** and the **True Risk**.

The authors have proved that this framework can be used to obtain generalization bounds for many algorithms, including the $k-$NN that we will present in the next section.

1.2 Parameters tuning and Evaluation of an hypothesis

Even though the generalization bounds studied in the previous section can be of great interest, *e.g.* to derive guarantees or to get insight into how to design new algorithms, they are often pessimistic and thus do not give an objective estimation of the quality of h. Since the joint distribution $\mathcal{D}_\mathcal{Z}$ is unknown, one has to figure out a solution to empirically estimate the generalization capacity of the hypothesis h by only using the training set S.

This latter is typically splitted into two subsets $S_1 \cup S_2$, one for learning (S_1) and one for testing and so evaluating the quality of the hypothesis (S_2). S_1 has to be used for both learning the parameters θ of the model and tuning the hyperparameters of the algorithm (*e.g.* the regularization parameter λ, the number k of neighbors in a $k-$NN classifier, the number of layers in a deep neural network, etc.). In order to have a fair process, S_1 is usually divided into two subsets, one for learning the parameters θ according to different sets of hyperparameters, the training set, and one for selecting the best hyperparameters, usually called the validation set. The global procedure is summarized in Figure 1.5

In order to be robust to sampling bias, the train/validation procedure can be repeated several times leading to the well known Cross-Validation method.

1.2.1 Cross validation

As illustrated in Figure 1.6, the principle is to separate the training set (the test set already kept apart) into k subsets of same size. Then, for each set of parameters that we want to test, we carry out k learning phases of the model, so that each of the k subsets is considered a test

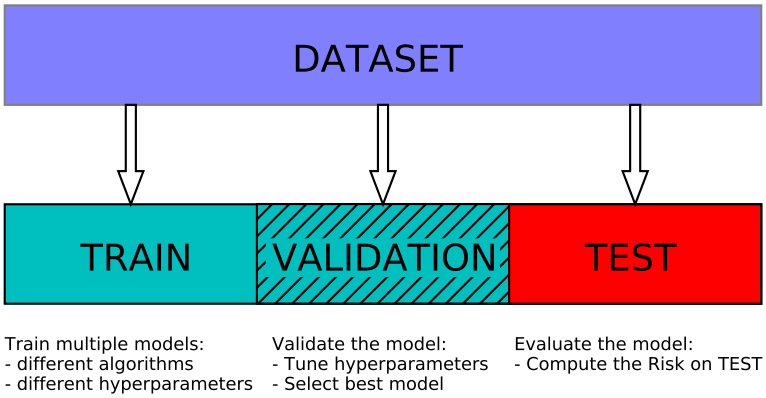

Figure 1.5: Illustration of the Train/Validation/Test procedure. In cyan, the set S_1 is split into 2 subsets. The first one (plained) used to train the model. The second one (hatched) used to validate the model. In red, the test set S_2 used to evaluate the model.

set. To do this, we select $k-1$ subsets and leave one apart to be considered as an unseen set. Thus, we learn on the set of $k-1$ selected parts, the training folds, and we evaluate the learned model on the set apart, the validation fold. Each of these k trainings therefore give k evaluation measures. Finally, we keep the set of parameters that will offer the best average evaluation over these k training phases.

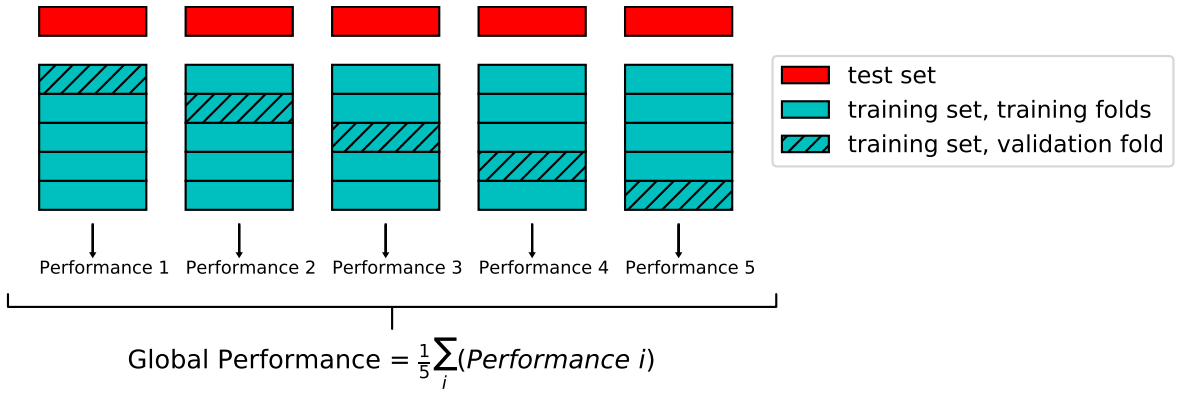

Figure 1.6: Illustration of a 5-folds cross validation. In cyan, the training set is split into 5 folds, used for cross validating. In red, the test set that will not be used during the cross validation. When all 5 folds have been used in validation, the average of the 5 performances will be taken to evaluate the global performance of the tested parameter set.

1.2.2 Performance measures

In order to evaluate the classification predictions of a learned model on test data, it is common to produce the Confusion Matrix. This is the summary of the counts per class according to the actual labels and the predictions. It allows one to see immediatly the capacity of the model to classify correctly, and when it fails, which labels are subject to more errors. In the case of a binary classification, the confusion matrix takes the form of the Table 1.1.

Table 1.1: Confusion Matrix in a binary classification setting.

		Predicted labels	
		$+1$	-1
True labels	$+1$	True Positive (TP)	False Negative (FN)
	-1	False Positive (FP)	True Negative (TN)

True Positive (TP) and True Negative (TN) correspond to cases where the model correctly classifies examples by recovering their actual positive and negative labels respectively. False Positive (FP) and False Negative (FN) correspond to two types of possible errors, also called false and missed alarms respectively. FP corresponds to negative examples wrongly predicted as positive and FN to positive examples wrongly predicted as negative.

Since it is often simpler to work with a single value, it is suitable to synthesize the information from this confusion matrix. There are several ways to do it and this will depend mainly on the context and the task at hand. The most direct measure to summarize the confusion matrix is the *Accuracy*, which measures the proportion of correct predictions and is defined as follows:

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN} \in [0, 1]$$

Note that the complementary of the *Accuracy* is called the *Error Rate* which corresponds to the quantity minimized by the $0/1$ loss.

Other performance measures coming from the Confusion Matrix exist and will be presented in Section 1.5.1 dedicated to the imbalanced data.

1.3 Popular Learning Algorithms

In this section, we present some of the most popular supervised learning algorithms. Among them, the $k-$NN classifier and Decision Trees will have an important place in this thesis as they are at the heart of our contributions.

1.3.1 $k-$Nearest Neighbor ($k-$NN)

The Nearest Neighbor algorithm [29] is probably the simplest supervised learning algorithm. Despite its simplicity, $k-$NN is theoretically well rooted as it can be shown that it approximates the Bayesian classifier and its error is upper bounded by twice the (smallest) Bayesian error. Following the saying 'Birds of a feather flock together', its principle is that an example should probably be of the same class as its nearest neighbor(s). Thus, given a training set S and a distance function $\mathcal{D}$, to label a new example $\mathbf{x}$, the algorithm computes the distances of $\mathbf{x}$ to the m examples in S to retain only the k closest ones. Once this subset of S is selected, a simple majority vote among the corresponding labels allows us to label the query (see Figure 1.7). One of the particularities of $k-$NN is that it does not really require learning. Indeed, once the set S is stored in memory, only calculations of distances are needed at test time to label $\mathbf{x}$. That is why $k-$NN is often called 'lazy algorithm'. The only important parameter is the number of neighbors considered which can be tuned by cross-validation.

Many adaptations of this algorithm have been proposed. For example, if $k > 1$, it is possible to use weights inversely proportional to the distances between the test point and its neighbors in order to determine its label [34]. In this case, the closer a training example is to the test example, the more influence it has on the predicted label. This is particularly useful for managing ties that often occur in majority votes. In Figure 1.8, we illustrate both the influence of k in the algorithm and the difference between the uniform majority vote and the distance-weighted version.

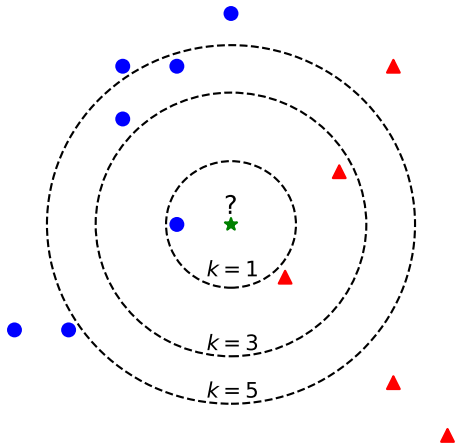

Figure 1.7: Illustration of the classification with a $k-$NN classifier. The blue circles and red triangles correspond to the training examples from two classes. The green star corresponds to the new example to label. If we consider $k = 1$ or $k = 5$ neighbors, it will be labeled as a blue circle while if we consider $k = 3$, it will be labeled as a red triangle.

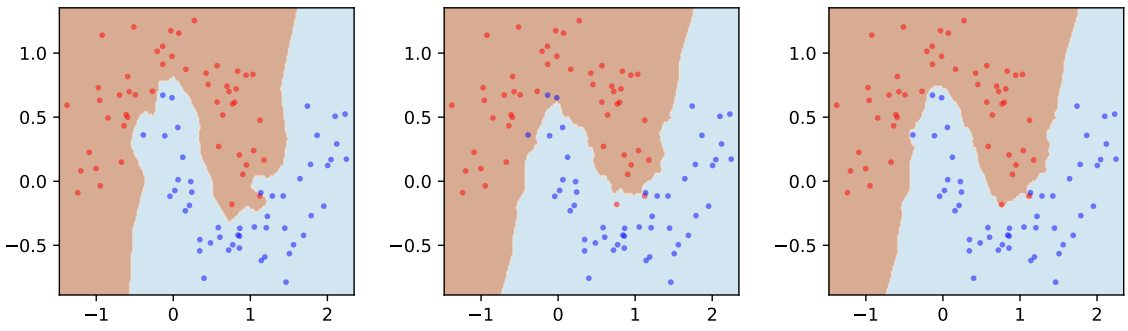

Figure 1.8: Impact of the number of considered neighbors and the distances in the $k-$NN algorithm. Here, the underlying distributions are two moons with a Gaussian noise of standard deviation 0.2. (Left) $k = 1$ with a uniform majority vote. The points of the training set are properly separated into two subsets corresponding to their labels. (Center) $k = 11$ with a uniform majority vote. Here, some points at the tips of the moons are influenced by the centers of the opposite moons and end up in the other classification area because the majority of their 11 neighbors have a different label. (Right) $k = 11$ with a distance-weighted vote. By taking into account the distances to the neighbors, the classifications areas become more appropriate to the distribution, although the border between the two remains a sensitive area.

An important point of this algorithm is the choice of the distance used. This is essential and has a great influence on the accuracy of the model. In the case of numerical data, the most commonly used is the Euclidean distance. However, other distances can be used, often related and adapted to the task to be solved, such as the Levenshtein distance for strings, trees and graphs, or the Dynamic Time Warping for time series. A whole branch of Machine Learning works on how to learn dedicated distances. This field is called Metric Learning [10]. Once these distances are learned, they can be directly used by the Nearest Neighbor algorithm. In this case, the performances of the $k-$NN are usually significantly improved. Metric Learning being related to the second contribution of this thesis, it will be detailed in Section 1.4.

1.3.2 Support Vector Machine (SVM)

Support Vector Machines were introduced in the mid-1990s from the work of Vladimir Vapnik on Statistical Learning Theory [12, 28, 112]. SVMs are a generalization of linear separators and are based on two key notions. The first one is the notion of margin which expresses the distance between the separation boundary and the nearest examples (called Support Vectors). SVM aims at maximizing the margin. The second key notion is that of kernel. Since the training data are not necessarily linearly separable, the idea is to find a space, of higher and possibly infinite dimension, in which the data are potentially linearly separable. Kernels allow this by transforming the inner products involved in the optimization problem in the high dimensional space into a simple function evaluation. This technique is called the kernel trick. In Figure 1.9, we can see an example of projection of a non linearly separable 2D toy example in a three dimensional space. In this 3D space, the data seems now to be almost separable by a horizontal plane.

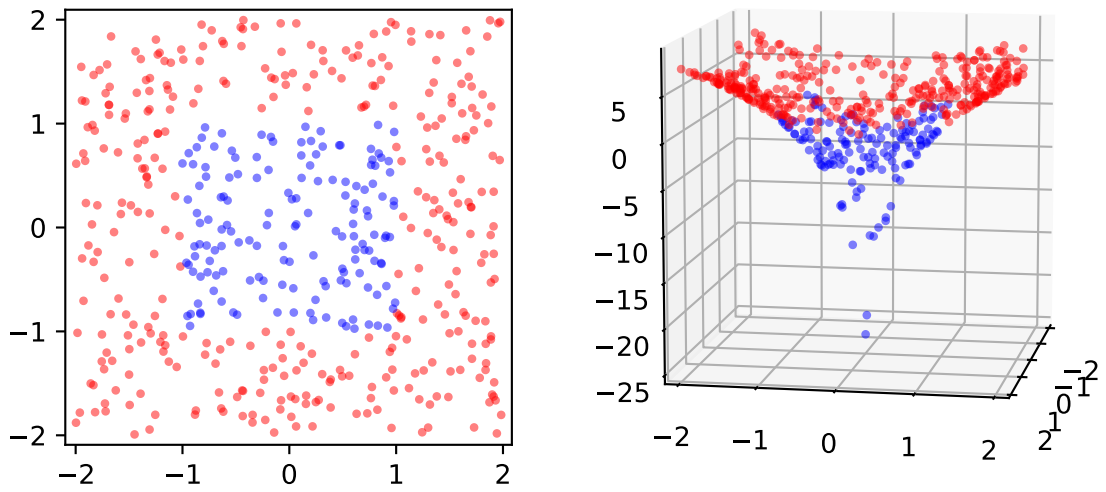

Figure 1.9: Illustration of the projection of data into a higher dimensional space. (Left) Our non linearly separable 2D toy dataset. (Right) Its projection in a 3D space with $f(x, x') = log((0.9 * \langle x, x' \rangle)^4)$ which almost allows a linear separation.

Linear separator To understand how a SVM works, let's start from the notion of linear separator. In a d-dimensional space $\mathbb{R}^d$, a linear separator h is of the form:

$$\begin{aligned} h : \mathbb{R}^d &\to \{-1, 1\} \\ \mathbf{x} &\mapsto h(\mathbf{x}) = sign(\langle \mathbf{w}, \mathbf{x} \rangle + b) \end{aligned}$$

with $\langle ., . \rangle$ the inner product, $\mathbf{w}$ a weight vector and b the bias.

h defines then a hyperplane of equation $\langle \mathbf{w}, \mathbf{x} \rangle + b = 0$ which separates $\mathbb{R}^d$ into two regions. This hyperplane can also be noted $(\mathbf{w}, b)$. The principle of SVM is then to learn a linear separator but in an appropriate feature space $\mathcal{F}$.

Large margin In an ideal case where the classes are perfectly separable, it would be possible to find many hyperplanes reaching a perfect accuracy. This is why [12] proposed to select the one that allows the largest margin between the separator h and the support vectors, that correspond in this case to the closest points to the hyperplane.

The margin of an instance $\mathbf{z}_i = (\mathbf{x}_i, y_i)$, defined as $\gamma_i = y_i \times (\langle \mathbf{w}, \mathbf{x}_i \rangle + b)$, is positive if $\mathbf{z}_i$ is correctly classified. And the margin γ of the hyperplane $(\mathbf{w}, b)$ with respect to the training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ is defined as:

$$\gamma = \min_{1 \leq i \leq m} \gamma_i = \min_{1 \leq i \leq m} y_i \times (\langle \mathbf{w}, \mathbf{x}_i \rangle + b).$$

By maximizing the margin between the separator and the support vectors, we obtain the primal formulation of Hard Margin SVM:

$$\begin{aligned} &\min_{\mathbf{w}, b} \frac{1}{2} ||\mathbf{w}||_2^2 \\ &\text{subject to } y_i(\langle \mathbf{w}, x_i \rangle + b) \geq 1, \text{ for } 1 \leq i \leq m. \end{aligned}$$

By using the Lagrange duality, we obtain the dual formulation of hard margin SVM:

$$\begin{aligned} &\max_{\boldsymbol{\alpha}} \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\ &\text{subject to } \sum_{i=1}^m y_i \alpha_i = 0 \text{ and } \boldsymbol{\alpha} \succeq 0. \end{aligned}$$

Kernel trick To find an adequate feature space $\mathcal{F}$ where a linear separator is learned, we have to use a transformation $\phi : \mathcal{X} \to \mathcal{F}$ and replace all $\langle \mathbf{x}, \mathbf{x}' \rangle$ by $\langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$ in the aforementioned problem.

The notion of Kernel can then help us to compute this inner product without having to explicitely apply ϕ. A function $K : \mathcal{X} \times \mathcal{X} \to [-1, 1]$ is a Kernel if there exists a function $\phi : \mathcal{X} \to \mathcal{F}$ such that $K(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$.

By using this Kernel trick on the hard margin dual formulation, we get the Kernelized SVM hard margin dual formulation:

$$\begin{aligned} &\max_{\boldsymbol{\alpha}} \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \\ &\text{subject to } \sum_{i=1}^m y_i \alpha_i = 0 \text{ and } \boldsymbol{\alpha} \succeq 0. \end{aligned}$$

The Kernelized separator h becomes:

$$h(\mathbf{x}) = \sum_{i \in SV} y_i \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b$$

with SV the set of support vectors.

Soft Margin SVM In practice, it is very unlikely that the data are perfectly linearly separable, even when projected into a very high dimensional space. Therefore, it is necessary to relax this constraint by accepting that some points violate this margin requirement (see Figure 1.10). This new version of the problem will thus correspond to optimizing a tradeoff between a large margin and the amount of violations of this margin. To do this, one adds slack variables to the primal problem, which allow again to switch to the dual problem and use the Kernel Trick.

The SVM soft margin primal formulation is defined as follows:

$$\begin{aligned}&\min_{\mathbf{w},b,\boldsymbol{\xi}} \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_{i=1}^{m}\xi_i\\&\text{subject to } y_i(\langle \mathbf{w}, x_i\rangle + b) \geq 1 - \xi_i, \text{ for } 1 \leq i \leq m\\&\text{and } \boldsymbol{\xi} \succeq 0\end{aligned}$$

with $\boldsymbol{\xi}$, the slack variables that allow some instances to violate the margin constraint and $C \geq 0$ the tradeoff parameter between the size of the margin and the magnitude of the violations.

This formulation is equivalent to the following unconstrained formulation:

$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_{i=1}^{m}[1 - y_i(\langle \mathbf{w}, x_i\rangle + b)]_+$$

where $[1-a]_+$ is the hinge loss.

Finally, the Kernelized SVM soft margin dual formulation is defined as follows:

$$\begin{aligned}&\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)\\&\text{subject to } \sum_{i=1}^{m} y_i\alpha_i = 0 \text{ and } 0 \leq \alpha_i \leq C, \text{ for } 1 \leq i \leq m\end{aligned}$$

1.3.3 Logistic Regression

Logistic Regression [11] is a popular statistical method for predicting the probability of the occurence of an event Y from a set of descriptive variables X, *i.e.* $P(Y = 1|X)$. By thresholding this probability (typically at 0.5), Logistic Regression can thus be easily used in Machine Learning for binary classification.

Let us consider the logistic (or sigmoïd) function:

$$\begin{aligned}s : \mathbb{R} &\to \{0, 1\}\\x &\mapsto s(x) = \frac{1}{1 + \exp(-x)}.\end{aligned}$$

Assume that we want to learn a function $h_\theta(\mathbf{x}) = s(\theta^T\mathbf{x})$ such that:

$$P(y = 1|\mathbf{x}) = h_\theta(\mathbf{x}) \text{ and } P(y = 0|\mathbf{x}) = 1 - h_\theta(\mathbf{x}).$$

So, for all $y \in \{0; 1\}$, we get:

$$P(y|\mathbf{x}) = h_\theta(\mathbf{x})^y \times (1 - h_\theta(\mathbf{x}))^{1-y}$$

and thus, the likelihood L and its log are defined as follows:

$$L(\theta) = \prod_i P(y_i|\mathbf{x}_i) = \prod_i h_\theta(\mathbf{x}_i)^{y_i} \times (1 - h_\theta(\mathbf{x}_i))^{1-y_i}$$

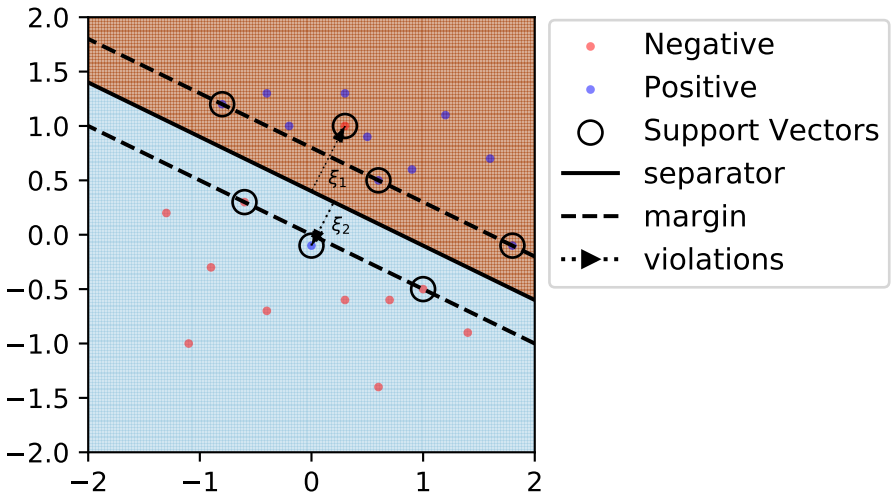

Figure 1.10: Illustration of a SVM separator on a toy example, with 2 points that violate the margin constraint. The black line corresponds to the linear separator. The dashed lines represent the margin. The circled points are the support vectors. Note that by definition of the solution of the optimization problem considered, the points leading to margin violations are also considered as support vectors.

and:

$$\log(L(\theta)) = \sum_i \left(y_i \log\left(h_\theta\left(\mathbf{x}_i\right)\right) + \left(1 - y_i\right) \log\left(1 - h_\theta\left(\mathbf{x}_i\right)\right)\right).$$

By maximizing the log of the likelihood with respect to θ, we can find the best parameter so that h_θ fits the best the data.

The Logistic Regression also corresponds to the estimation of the logarithm of the odds, i.e. the logarithm of the ratio of probabilities, by a linear model $\tilde{h}_\theta(\mathbf{x})$:

$$\log(odds) = \log\left(\frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})}\right) \approx \tilde{h}_\theta(\mathbf{x}).$$

Once this linear model $\tilde{h}_\theta(\mathbf{x})$ has been learned, we can directly obtain the probability of the event occurring with the logistic function:

$$P(y=1|\mathbf{x}) = \frac{\exp\left(\tilde{h}_\theta(\mathbf{x})\right)}{1+\exp\left(\tilde{h}_\theta(\mathbf{x})\right)} = \frac{1}{1+\exp\left(-\tilde{h}_\theta(\mathbf{x})\right)}.$$

In Figure 1.11, we can find an illustration of the logistic curve that best fits the data. The corresponding parameter is $\theta^T \simeq (10.173; -22.520)$ and thus, the logistic function is

$$h_\theta(\mathbf{x}) = \frac{1}{1+\exp(-(10.173\mathbf{x} - 22.520))}.$$

With a threshold of 0.5, all points with $\mathbf{x} < 2.224$ will be labelled as negative and all points with $\mathbf{x} \geq 2.224$ will be labelled as positive. In this example, the log of the odds is linearly estimated with $\tilde{h}_\theta(\mathbf{x}) \approx 10.173\mathbf{x} - 22.520$.

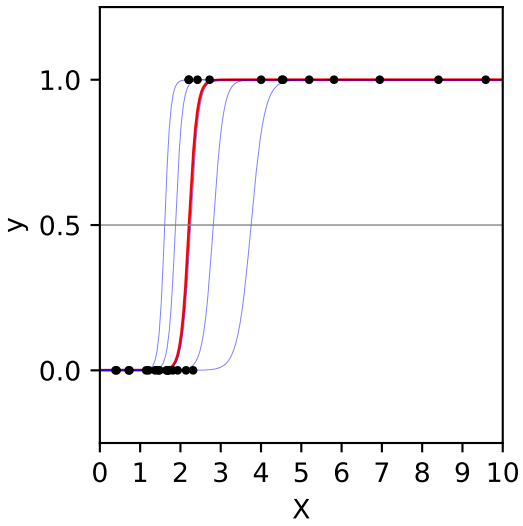

Figure 1.11: Example of Logistic Regression on a toy example. (Blue curves) Example of logistic functions with different θ parameters. (Red curve) Best logistic function, learned from the data. (Grey line) The threshold. All points of the red curve that are above the threshold will be labelled 1 while all points below the threshold will be labelled 0.

1.3.4 Decision Trees

Decision Trees (DT) [72, 90, 91] are among the simplest and most widely used methods in Machine Learning. In their basic form, DT are binary trees that recursively split from a root the dataset according to some criterion (*e.g.* Information Gain or Gini Index) until we are left with pure leaf nodes, that is nodes with examples coming from only one class. Once learned, DT can be expressed in terms of decision rules (from the root to the leaves) that can be used to predict the label of new examples.

Figure 1.12 shows an example of DT (left) learned from a toy dataset (right). The condition in the root node splits the dataset in two groups of examples, those with x_1 less than or equal to -0.986 and the others. This split, represented by the blue line in the Figure on the right, allows us to have a pure leaf on the left, because all the examples satisfying the condition have the same label, and a new node on the right where the examples inside do not all have the same label. A new decision rule ("x_1 less than or equal to 0.996" corresponding to the purple line) is then created for the examples in this node to separate them. And so on, until the data are clearly separated into subsets of the same label. Here we have obtained five pure leaves in the tree, which correspond to the five regions of the space $\mathbb{R}^2$, obtained from the hyperplanes corresponding to the decision rules.

A DT algorithm partitions the space $\mathbb{R}^d$ into regions in which training examples share some characteristics and often, identical labels. In order to predict the label of a new example, we feed it to the DT, following the corresponding path and assign the label of the resulting leaf.

In order to prevent some overfitting phenomenon, pruning stategies can be applied to reduce the size of the DT and avoid the induction of leaves containing too few examples.

Although DT can be viewed as a simple set of if-then-else statements, note that the decision rules are learned from the training set in an optimal way. To do this, we typically calculate the impurity (or uncertainty) of a node. It can be done via the Shannon Entropy [101, 90, 91], which measures the information contained in a node and defined as follows:

$$E(node) = \sum_i - p_i \log(p_i)$$

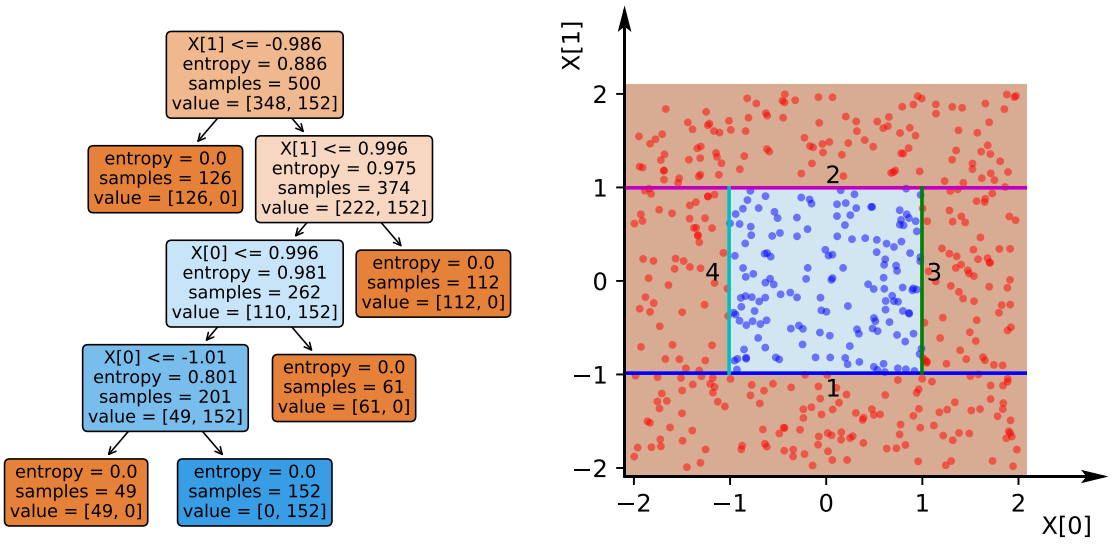

Figure 1.12: Example of Decision Tree (left) and the corresponding classifier induced by this DT on a toy dataset (right). Each internal node contains the decision rule linking a feature to a value, the entropy of the node, the number of examples in the node, as well as the distribution of classes [red, blue].

with p_i the probability of the class i estimated from the subset of examples falling in the node.

If this Entropy is high, the prediction is uncertain. On the contrary, pure leaves have zero Entropy. The Information Gain IG of a potential split of a parent node is calculated by subtracting the entropies of its child nodes, weighting by the rate of examples in each child node. More formally,

$$IG(split) = E(Parent) - \sum_j w_j E(Child_j).$$

This calculation is done for all features and all possible splits on these features and the best couple feature/split, that maximizes the Information Gain is selected. An illustration is given in Figure 1.13. We consider a node that contains 5 blue and 5 red examples, so with an entropy equal to 1. The computation of the Information Gain of each possible split leads us to select the 6th split. It will split the dataset into a right node containing the 5 red examples and 1 blue example while the left node will be pure with 4 blue examples. With this split, the entropy of the left part will be 0.65 and that of the right part 0, which leads to an Information Gain of 0.61 which is the maximum gain.

Another way to calculate the impurity of a node is to use the Gini Index [21, 72] instead of the Information Gain. As with the latter, if the Gini Index G is high, the impurity of the node is large and G equals zero if the node is pure. The definition of the Gini Index is:

$$G = 1 - \sum_i p_i^2.$$

The advantage of the Gini Index over the Information Gain is that it is less expensive to compute as it does not involve the costly calculation of the logarithm.

In Figure 1.14, we illustrate two different classifiers learned with the two different impurity criteria on a toy dataset.

Note that a Decision Tree algorithm is a greedy method. It selects the current best split that maximizes the Information Gain without backtracking and challenging a previous split. So, there is no guarantee to obtain the global optimal solution. On the other hand, it is worth

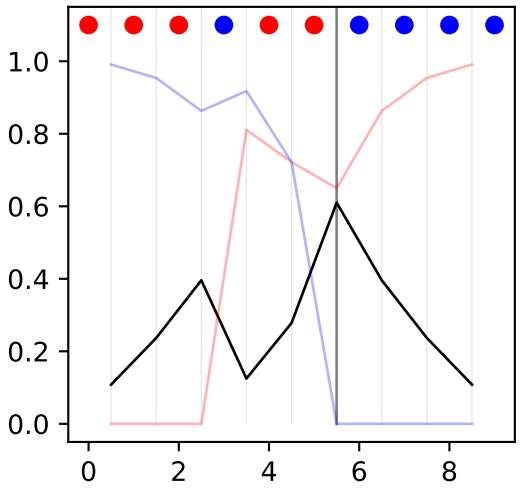

Figure 1.13: Example of the splitting rule with the Information Gain on a toy example. (Top line) Projection of the points of a set along one of the axes. (Vertical lines) Lists of possible splits. In bold, the best split. (Red/blue curve) Entropy of the left/right part of the split. (Black curve) Information Gain corresponding to the split.

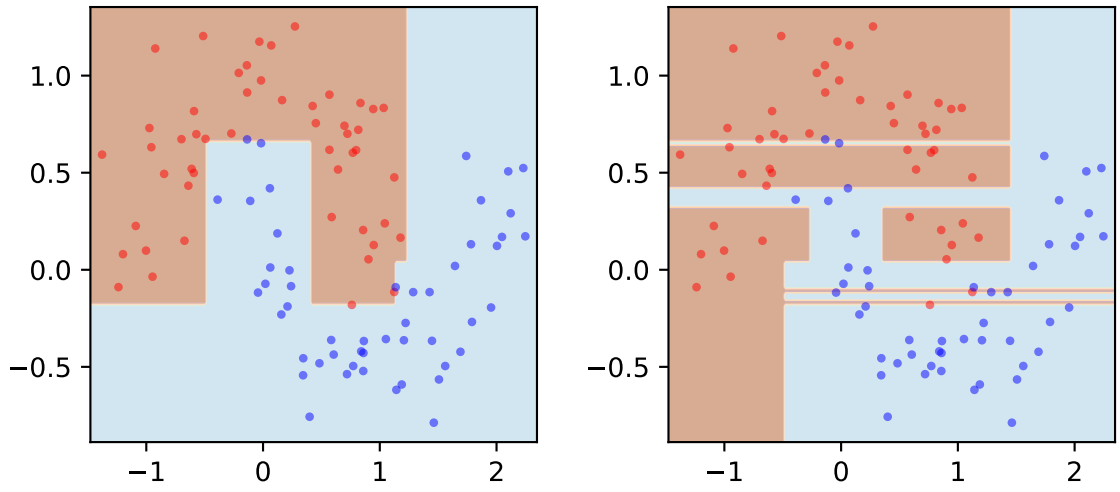

Figure 1.14: Example of classifiers induced by a DT with Information Gain (left) and Gini Index (right) on a toy example.

noting that this method is very fast and achieves very good results despite its simplicity. This explains why it is widely used. Another important point about Decision Trees is that they are inherently interpretable by construction. We will benefit from this advantage in Chapter 4.

1.3.5 Ensemble Learning

Ensemble methods are learning algorithms that construct a set of n hypotheses $\{h_i\}_{i=1}^{n}$ whose individual predictions are combined in some way to classify new examples. Each of the hypotheses must be accurate, *i.e.* at least better than random guessing, and diverse, *i.e.* they must be complementary by making errors on different examples. It has been shown that such ensemble methods allow one to reduce the bias and variance. They are derived from the 'Wisdom of the Crowd' principle: A crowd of amateurs is more often right than a single expert. The law of large numbers explains this principle. As the number of individuals increases, the performance of the majority approaches 100% of correct answers, as long as each individual does better than 50% correct and there is diversity among the individuals. It is this diversity that makes for strength in numbers. The crowd must therefore be large, competent and diverse. To make the final prediction, the individual decisions of the base classifiers can be merged with a meta-learner, averaged, or weighted according to certain criteria.

There are many ways to achieve diversity in classifiers, by training:

- different kinds of classifiers on the same training set S, such as a mixture of $k-$NN, SVM, DT, etc.
- the same type of classifiers on the same training set S but with different hyperparameters, such as different kernels for the SVM or different numbers of neighbors k for the $k-$NN.
- the same classifier but varying the training set, by some sampling scheme.

The first two categories correspond to *heterogeneous ensemble methods* because it is the algorithm that changes and not the distribution of points in the training set. These methods include *Stacked Generalization* (also named Stacking) [125], for example. The third category corresponds to *homogeneous ensemble methods* because the algorithm remains intrinsically the same, while the distribution of the points is modified to create diversity. Different methods fall into the scope of this family, like *Boostrap Aggregating* (also named *Bagging*) [15], Random Forests [16, 59] or *Boosting* [46, 47, 48, 80].

We briefly introduce Stacking and Bagging below before entering more into details of Boosting in the next Section.

Stacking The idea behind Stacking is to train an ensemble of n different classifiers on the training set S of size m. Then, m vectors of size n are generated from the decisions of the n classifiers, one per example in S. Finally, a combiner, or meta-learner, is learned on these n predictions to predict the final decision (see Figure 1.15). Any algorithm can be used as a meta-learner although Logistic Regression has been shown to be very efficient in practice.

Bagging The idea in Bagging is to create, in parallel, several models learned from random subsets of the training set S. To do this, we use a sampling technique called Bootstrapping, which consists in replacing, after each draw, the selected data into the original dataset. The learned models are thus diverse since they are learned on slightly different subsets. Once this set of models has been learned, a majority vote can be applied to determine the final prediction of the overall model (see Figure 1.16).

Note that bagging can be efficiently used to train Random Forests [16, 59], *i.e.* a set of DT learned from various distributions of examples and features generated by sampling.

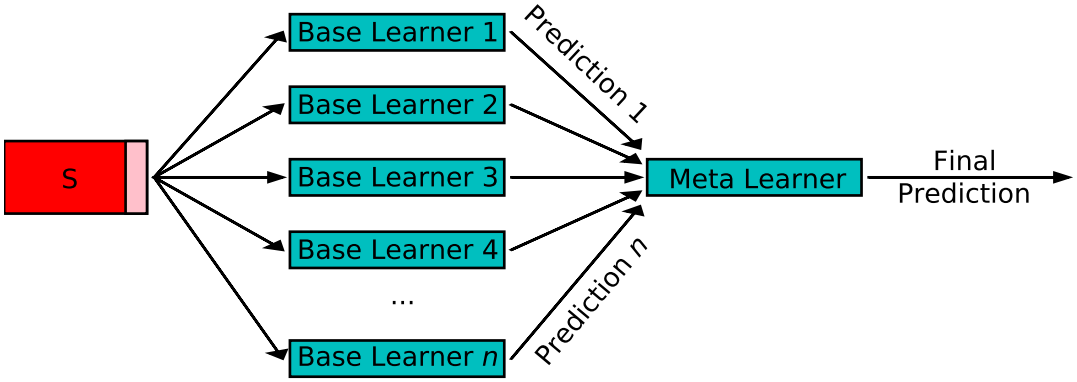

Figure 1.15: Illustration of the stacking of n base learners, trained on the training set S (red). The n predictions on the test set (pink) are combined by the meta learner to have the final prediction.

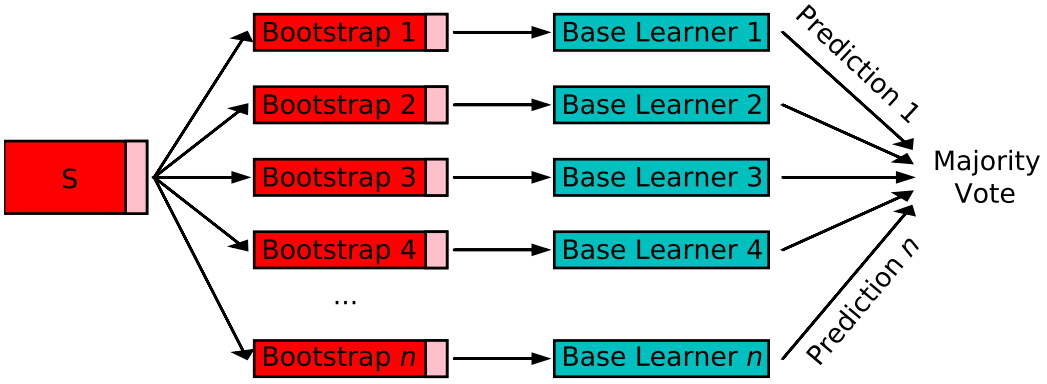

Figure 1.16: Illustration of bagging. n subsamples are generated from the dataset S by bootstrapping. Then, n base learners of the same type are trained on these subsets. A majority vote is applied on their predictions to have the final prediction.

Algorithm 1.1: AdaBoost Algorithm

Input: a learning sample S, a number of iterations T, a weak learner h
Output: a global strong learner $\mathcal{H}$

for *i from 1 to m* **do**
  $W_1(\mathbf{x}_i) = 1/m$ *// Initialization of the weights of the examples*
end
for *t from 1 to T* **do**
  $h_t = h(S, W_t)$ *// Training of the weak learner h_t on the weighted dataset*
  $\hat{\epsilon}_t = \sum_{\mathbf{x}_i \text{ s.t. } y_i \neq h_t(\mathbf{x}_i)} W_t(\mathbf{x}_i)$ *// Computation of the weighted empirical error of h_t*
  $\alpha_t = \frac{1}{2} \ln \frac{1-\hat{\epsilon}_t}{\hat{\epsilon}_t}$ *// Computation of the weight of h_t*
  for *i from 1 to m* **do**
    $W_{t+1}(\mathbf{x}_i) = W_t(\mathbf{x}_i) \times \exp(-\alpha_t y_i h_t(\mathbf{x}_i))/N_t$ *// Update of the weights of the examples*
    // With N_t a normalization coefficient
  end
end
$f(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$;
return $\mathcal{H}_T : \mathbf{x} \mapsto \mathcal{H}_T(\mathbf{x}) = sign(f(\mathbf{x}))$;

1.3.6 Boosting

Boosting [98] is an ensemble method which aims at building a strong model by combining so-called weak hypotheses that are learned step by step. The principle here is to have each new model to correct the errors of the previous one by changing the distribution of the training data in favor of the 'hard' examples.

The most popular boosting algorithms are AdaBoost [46], for Adaptive Boosting, and Gradient Boosting [47, 48, 80].

AdaBoost In AdaBoost, each new classifier corrects the errors of the previous one by increasing the weight of the misclassified examples so that it will focus more on them. AdaBoost assigns a coefficient to each weak learner in an optimal way, *i.e.* the weight which minimizes the training error. The pseudo-code of AdaBoost is presented in Algorithm 1.1. Starting from a uniform distribution of the data, a new hypothesis h_t is learned. Then, the distribution is updated as follows:

$$W_{t+1}(\mathbf{x}_i) = W_t(\mathbf{x}_i) \times \frac{\exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{N_t}$$

where α_t corresponds to the weight that will be given to the current hypothesis h_t in the final model and N_t is a normalization coefficient. After T iterations, the different learned classifiers h_t are linearly combined to obtain the final classifier $\mathcal{H}_T$. An illustration of AdaBoost is shown in Figure 1.17.

AdaBoost is fast and simple. It comes with many theoretical guarantees, not only on the training error but also on the generalization risk. In practice, it works very well with decision stumps (DTs of depth 1), but is subject to overfitting in the presence of outliers.

Gradient Boosting For the past few years, Gradient Boosting has become one of the most widely used algorithms, especially when working with large amounts of data. An evidence of its popularity can be illustrated by the list of winners of the big data science competitions, as Kaggle, or the number of hits from search engines for XGBoost [25], one of the currently most efficient Gradient Boosting algorithm (already used at the DGFiP), or LightGBM [62].

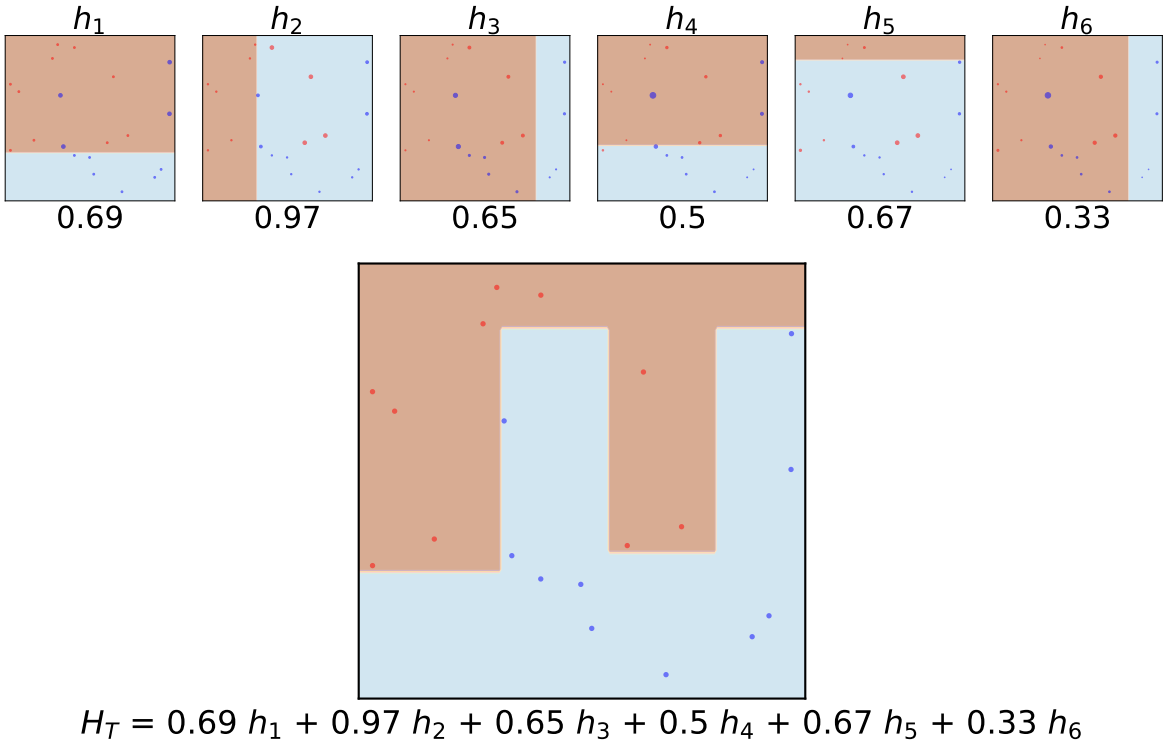

Figure 1.17: Example of 6 iterations of AdaBoost on a toy example when using decision stumps as weak hypotheses. (Top) The 6 iterations with the corresponding weights α_t. (Bottom) The final classifier as a weighted sum of the 6 decision stumps.

Like AdaBoost, Gradient Boosting is an ensemble learning method where a combination of weak classifiers is improved by adding a new element to it, which corrects the errors of the current ensemble by learning from the residuals, expressed in the form of gradients. Unlike AdaBoost which is based on the exponential loss (see Algorithm 1.1), Gradient Boosting can use any differentiable loss function.

The pseudo-code is presented in Algorithm 1.2. It starts by learning a first model H_0 on the training set S. Once this model is learned, two important steps are repeated T times. The first one computes the pseudo-residuals r_i, which correspond to the error of the model on the training set, using the gradient of the chosen loss. Once these pseudo-residuals are calculated, a new weak classifier h is trained to predict them and is added to the global model so that the combination of the two performs better than the previous model. This new weak classifier is weighted by an optimal coefficient γ_t and a learning rate α.

Algorithm 1.2: Gradient Boosting Algorithm

Input: a learning sample S, a number of iterations T, a weak learner h, a differentiable Loss Function $l(y, H(\mathbf{x}))$ and a learning rate $\alpha \in]0; 1]$

Output: a final model $H_T(\mathbf{x})$

$H_0(\mathbf{x}) = \arg\min_{\gamma} \sum_{i=1}^{m} l(y_i, \gamma) = h(x_i, 1/|S|_{i=1}^{m})$ // *Initialization of the model*

for *t from 1 to T* **do**

  for *i from 1 to m* **do**

    $r_{i,t} = -\left[\frac{\partial\, l(y_i, H(x_i))}{\partial H(x_i)}\right]_{H(\mathbf{x})=H_{t-1}(\mathbf{x})}$ // *Computation of the pseudo-residuals*

  end

  $h_t = h(\{\mathbf{x}_i, r_{i,t}\}_{i=1}^{m})$ // *Training of the weak learner h_t on the pseudo-residuals*

  $\gamma_t = \arg\min_{\gamma} \sum_{i=1}^{m} l(y_i, H_{t-1}(\mathbf{x}_i) + \gamma h_t(\mathbf{x}_i))$ // *Computation of the multiplier γ_t*

  $H_t(\mathbf{x}) = H_{t-1}(\mathbf{x}) + \alpha\gamma_t h_t(\mathbf{x})$ // *Update of the model*

end

return $H_T(\mathbf{x})$;

Following the introduction of the basics of Machine Learning and the main algorithms used in supervised learning for classification, it seems important to introduce another key concept, Metric Learning, which is at the heart of our third chapter. The Metric Learning is a field of Machine Learning that allows us to learn metrics that are better adapted to the algorithms and problems we are interested in. We introduce it in the next section.

1.4 Metric Learning

In Machine Learning, the notion of distance/metric plays an important role, particularly in classification algorithms such as SVM or $k-$NN, but also for ranking algorithms or unsupervised clustering. A metric can be used on different types of data, numerical or more structured like words or graphs. The problem with standard metrics is that they often fail to capture the idiosyncrasies of the application at hand because they are not parameterized. Metric Learning [10] is the field of Machine Learning that consists of learning a distance adapted to the application and that allows one to enhance classification algorithms. The basic idea of Metric Learning is to learn a transformation that assigns a small distance to pairs of similar examples while assigning a large distance to pairs of dissimilar samples, as illustrated in Figure 1.18. This implicitly induces a change in the representation space so that it can satisfy these constraints.

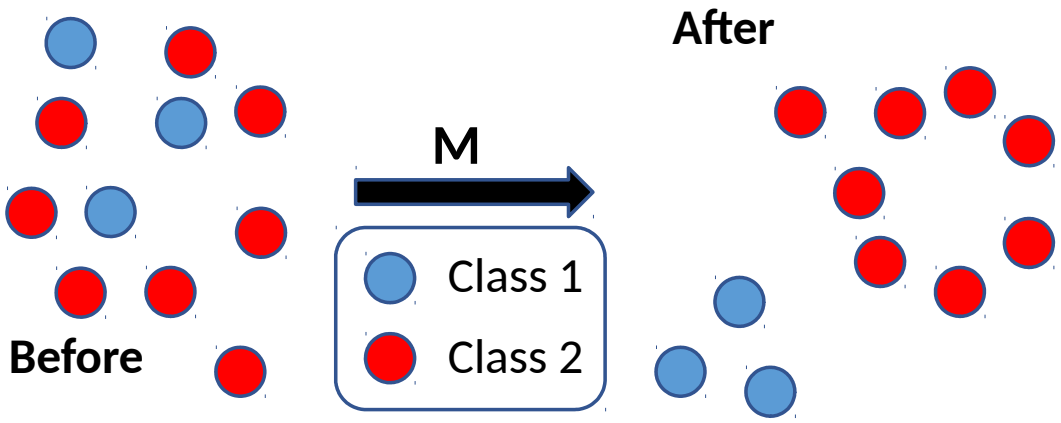

Figure 1.18: Illustration of the concept of Metric Learning.

Distances To be a true distance the learned metric has to satisfy the following conditions:

Definition 2. *A distance* $\mathcal{D} : X \times X \to \mathbb{R}^+$ *is a positive function such that the following conditions must be satisfied for any* $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \in X^3$*:*

- *non-negativity:* $\mathcal{D}(\mathbf{x}_1, \mathbf{x}_2) \geq 0$
- *identity of indiscernibles:* $\mathcal{D}(\mathbf{x}_1, \mathbf{x}_2) = 0 \Leftrightarrow \mathbf{x}_1 = \mathbf{x}_2$
- *symmetry:* $\mathcal{D}(\mathbf{x}_1, \mathbf{x}_2) = \mathcal{D}(\mathbf{x}_2, \mathbf{x}_1)$
- *triangle inequality:* $\mathcal{D}(\mathbf{x}_1, \mathbf{x}_3) \leq \mathcal{D}(\mathbf{x}_1, \mathbf{x}_2) + \mathcal{D}(\mathbf{x}_2, \mathbf{x}_3)$.

Among the metrics that can be learned, the Mahalanobis distance has received a tremendous interest during the past decade:

Definition 3. *A Mahalanobis distance* $\mathcal{D}_{\mathbf{M}}$ *is defined as follows:*

$$\forall (\mathbf{x}, \mathbf{x}') \in \left(\mathbb{R}^d\right)^2, \mathcal{D}_{\mathbf{M}}(\mathbf{x}, \mathbf{x}') = \sqrt{(\mathbf{x} - \mathbf{x}')^T \mathbf{M} (\mathbf{x} - \mathbf{x}')}$$

with $\mathbf{M} \in \mathbb{R}^{d \times d}$ *is a Positive Semi-Definite(PSD) matrix (*$\mathbf{M} \succeq 0$*).*

Note that when $\mathbf{M} = \mathbf{I}$, we recover the Euclidean distance. The advantage of these distances comes from a property of the PSD matrices which allows a decomposition of the matrix $\mathbf{M}$: If $\mathbf{M} \succeq 0$ then:

- $\forall \mathbf{x} \in \mathbb{R}^d, \mathbf{x}^T \mathbf{M} \mathbf{x} \geq 0$
- $\exists \mathbf{L} \in \mathbb{R}^{k \times d}$ such that $\mathbf{M} = \mathbf{L}^T \mathbf{L}$, with k the rank of the matrix $\mathbf{M}$.

Indeed, using this property, we notice that a Mahalanobis distance corresponds to the Euclidean distance after a linear projection of the data by the matrix $\mathbf{L}$ in a k dimensional space.

$$\begin{aligned}
\mathcal{D}_{\mathbf{M}}(\mathbf{x}, \mathbf{x}') &= \sqrt{(\mathbf{x} - \mathbf{x}')^T \mathbf{M} (\mathbf{x} - \mathbf{x}')} \\
&= \sqrt{(\mathbf{x} - \mathbf{x}')^T \mathbf{L}^T \mathbf{L} (\mathbf{x} - \mathbf{x}')} \\
&= \sqrt{(\mathbf{L}\mathbf{x} - \mathbf{L}\mathbf{x}')^T (\mathbf{L}\mathbf{x} - \mathbf{L}\mathbf{x}')} \\
&= \sqrt{(\tilde{\mathbf{x}} - \tilde{\mathbf{x}}')^T (\tilde{\mathbf{x}} - \tilde{\mathbf{x}}')} \\
&= \mathcal{D}_{euc}\left(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}'\right).
\end{aligned}$$

Problem formulation Based on the initial idea of grouping together similar examples and separating dissimilar ones, one can define three types of constraints:

$\mathcal{S} = \{(\mathbf{x}, \mathbf{x}')$: $\mathbf{x}$ and $\mathbf{x}'$ should be similar (must-link constraints)$\}$

$\mathcal{D} = \{(\mathbf{x}, \mathbf{x}')$: $\mathbf{x}$ and $\mathbf{x}'$ should be dissimilar (cannot-link constraints)$\}$

$\mathcal{R} = \{(\mathbf{x}, \mathbf{x}', \mathbf{x}'')$: $\mathbf{x}$ should be more similar to $\mathbf{x}'$ than to $\mathbf{x}''$(relative constraints)$\}$

Given these sets, a Mahalanobis distance problem typically takes the following form:

$$\mathbf{M}^* = \arg\min_{\mathbf{M} \succeq 0} l(\mathbf{M}, \mathcal{S}, \mathcal{D}, \mathcal{R}) + \lambda R(\mathbf{M})$$

with:

- $l(\mathbf{M}, \mathcal{S}, \mathcal{D}, \mathcal{R})$ a loss function that penalizes violated constraints,
- $R(\mathbf{M})$ a regularization term on $\mathbf{M}$,
- $\lambda \geq 0$ a regularization parameter.

The choice of the loss function and the regularization term will depend on the problem. This is also what differentiates the different state-of-the-art methods in Metric Learning. For the regularization term, it is quite common to take the Frobenius norm of the matrix $\mathbf{M}$, $||\mathbf{M}||_{\mathcal{F}}^2 = \sum_{i=1}^{d}\sum_{j=1}^{d} M_{ij}^2$.

Among the best known Metric Learning algorithms, one can cite LMNN [123] for Large Margin Nearest Neighbor. In their paper, the authors define $\mathcal{S}$ in a local way, thanks to $k-$NN. The k nearest neighbors, called "target neighbors", must be of the same class while the examples of the other classes, called "impostors", must be distant. The constraint sets can be written as follows:

$$\mathcal{S} = \{(\mathbf{x}, \mathbf{x}')\text{: } y = y' \text{ and } \mathbf{x}' \text{ is one of the } k\text{-}NN \text{ of } \mathbf{x}\}$$
$$\mathcal{R} = \{(\mathbf{x}, \mathbf{x}', \mathbf{x}'')\text{: } (\mathbf{x}, \mathbf{x}') \in \mathcal{S}, y \neq y''\}.$$

The formulation of the problem then becomes:

$$\min_{\mathbf{M} \succeq 0, \xi \geq 0} \sum_{(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{S}} \mathcal{D}^2_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) + \lambda \sum_{i,j,k} \xi_{ijk}$$

$$\text{such that } \mathcal{D}^2_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_k) - \mathcal{D}^2_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) \geq 1 - \xi_{ijk} \quad \forall (\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_j) \in \mathcal{R},$$
$$\text{and where } \xi_{ijk} \text{ are slack variables.}$$

This optimization problem expresses the idea that two points of the same class have to be closer to each other than to a sample of opposite class.

Another popular Metric Learning algorithm is ITML [30] for Information-Theoretic Metric Learning. The big difference with LMNN is the way to keep $\mathbf{M}$ in the set of Positive Semi-Definite matrices, a very expensive step in $\mathcal{O}(d^3)$. While LMNN projects during the gradient descent the update of $\mathbf{M}$ on the cone of PSD matrices at each step, ITML uses the LogDet divergence as a regularizer to achieve this goal, which is only quadratic in d.

The problem formulation is defined as follows:

$$\min_{\mathbf{M} \succeq 0, \xi \geq 0} trace(\mathbf{M}\mathbf{M}_0^{-1}) - \log\det(\mathbf{M}\mathbf{M}_0^{-1}) - d + \lambda \sum_{i,j} \xi_{ij}$$

$$\text{such that } \mathcal{D}^2_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) \leq u + \xi_{ij} \quad \forall (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{S}$$
$$\mathcal{D}^2_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) \leq v - \xi_{ij} \quad \forall (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{D}$$

with u and v two thresholds and $\mathbf{M}_0$ an initial PSD matrix, often the identity $\mathbf{I}$ or the inverse of the covariance matrix $\mathbf{\Sigma}^{-1}$.

1.5 Specificities of Imbalanced Learning

All of the above algorithms perform well in a standard classification framework if the data is balanced, *i.e.* with approximately as many positives as negatives. However, when it comes to fraud detection, this assumption does not hold anymore. Indeed, in this case, the class of interest, *i.e.* the fraudsters, is very largely under-represented compared to the majority negative class (the non-fraudsters).

In order to measure this imbalance, it is common to specify the minority/positive ratio (MR) in the dataset, or the imbalance ratio (IR) *i.e.* the number of negatives over the number of positives. They are respectively defined as follows:

$$MR = \frac{m^+}{m^+ + m^-} \text{ and } IR = \frac{m^-}{m^+}$$

As a result of this imbalance, all the notions we have seen, including the performance measures and the generalization guarantees, are challenged in this new setting. Different strategies can be applied to address this problem.

The imbalance can be taken into account by (i) resorting to more suitable performance measures, (ii) preprocessing the dataset to make it artificially balanced, (iii) reweighting the examples or (iv) directly considering the imbalance during the learning process. These four options are detailled in the following subsections.

1.5.1 Performance Measures

As explained previously, standard Machine Learning algorithms optimize the accuracy, or one of its surrogate versions. However, in an imbalanced context such as fraud detection the accuracy is not adapted. Let us take the example of a dataset with 99% of negative examples and 1% of positive samples that the algorithm is looking for as data of interest. If we take a model that always predicts the negative label, we get an *Accuracy* of 0.99 while missing all the positive examples. It is therefore important to optimize a more relevant measure in this context. Note that all the measures that will be presented in this section will take their values between 0 and 1 and the higher the value the better the model.

Precision* and *Recall *Precision* and *Recall* focus on the positives detected by the model. The former, also called $Positive\ Predictive\ Value$, is defined as the ratio:

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

Intuitively, this corresponds to the capacity of the model to not classify a negative example as positive.

On the other hand, *Recall*, also called *Sensitivity* or $True\ Positive\ Rate$, is defined as follows:

$$Recall = \frac{TP}{TP + FN}.$$

It measures the capacity of the model to retrieve positives.

F_1-score As the two previous measures focus on the True Positives in complementary ways, it may be interesting to synthesize their respective information into one measure. This is what the F_1-*score* (F_1, also called F_1-$Measure$) does by taking the harmonic mean of the $Precision$ and *Recall*:

$$F_1\text{-}score = 2 \times \frac{Precision \times Recall}{Precision + Recall} = \frac{TP}{TP + \frac{1}{2}(FP + FN)}.$$

In this case, *Precision* and *Recall* are considered homogeneously. More generally, the F_β-*score* (F_β) can be used to give more importance to one of the two measures, and find a good compromise between them.

$$F_\beta\text{-}score = (1+\beta^2) \times \frac{Precision \times Recall}{\beta^2 \times Precision + Recall}.$$

In this case, $\beta > 1$ will be used to give more weight to the *Recall* (this latter being considered β times as important as the *Precision*).

Balanced Accuracy Another measure that can be used in this setting is the *Balanced Accuracy* (BA) which avoids a too optimistic estimation of the performance of a model when classes are not equivalently represented. It corresponds to the average of the *Recall* scores per class, and thus, in binary classification, it corresponds to the arithmetic mean of the True Positive Rate (TPR) and True Negative Rate (TNR).

$$BA = \frac{1}{2}\left(\frac{TP}{TP+TN} + \frac{TN}{TN+FP}\right).$$

Note that if the dataset is balanced, BA is equivalent to the *Accuracy*. But in case of imbalance, BA will be more appropriate than *Accuracy* for model optimization since it will optimize *Recall* on each class simultaneously, and not just the overall rate of correct answers. Then, BA is interesting when you want to predict both the negative and positive class.

G-Measure* and *G-Mean Two other measures may be of interest in the imbalanced context even if they are less used in practice. Unlike the F_1-*score* which uses the harmonic mean to synthesize the information of *Precision* and *Recall*, the *G-Measure* is based on the geometric mean of these measures.

On the other hand, the *G-mean* takes the geometric mean of the True Positive and True Negative Rates, thus making the parallel with the *Balanced Accuracy*.

Their respective formulas are the following:

$$G\text{-}Measure = \sqrt{Precision \times Recall} \quad \text{and} \quad G\text{-}Mean = \sqrt{\frac{TP}{TP+TN} \times \frac{TN}{TN+FP}}$$

AUC-ROC An important and widely used measure in the imbalanced context is the Area Under the Receveir Operator Characteristic Curve (*AUC-ROC*), especially when looking for a ranking of the data. The *ROC* curve (Figure 1.19) is the curve of the True Positive Rate (the *Recall*) versus the False Positive Rate, *i.e.* $FP/(TN+FP)$. To build this curve, it is necessary to use the predicted probabilities instead of the predicted labels in order to obtain a hierarchy in the data. This is why *AUC-ROC* is often used in ranking algorithms. To get a good *AUC*, one wants the *ROC* curve to be as close as possible to the top left corner. This corresponds to a high True Positive Rate with a low False Positive Rate.

AUC-ROC can be computed as follows:

$$AUC\text{-}ROC = \frac{1}{m^+ \times m^-} \sum_{i=1}^{m^+} \sum_{j=1}^{m^-} \mathbf{I}_{0.5}(f(x_i^+) - f(x_j^-)),$$

where x_i^+ is the i^{th} positive sample and x_j^- is the j^{th} negative example, f is the scoring function assigning a probability to be positive, and $\mathbf{I}_{0.5}$ is an indicator function equal to 1 when $f(x_i^+) - f(x_j^-) > 0$, $\frac{1}{2}$ when $f(x_i^+) - f(x_j^-) = 0$ and 0 otherwise.

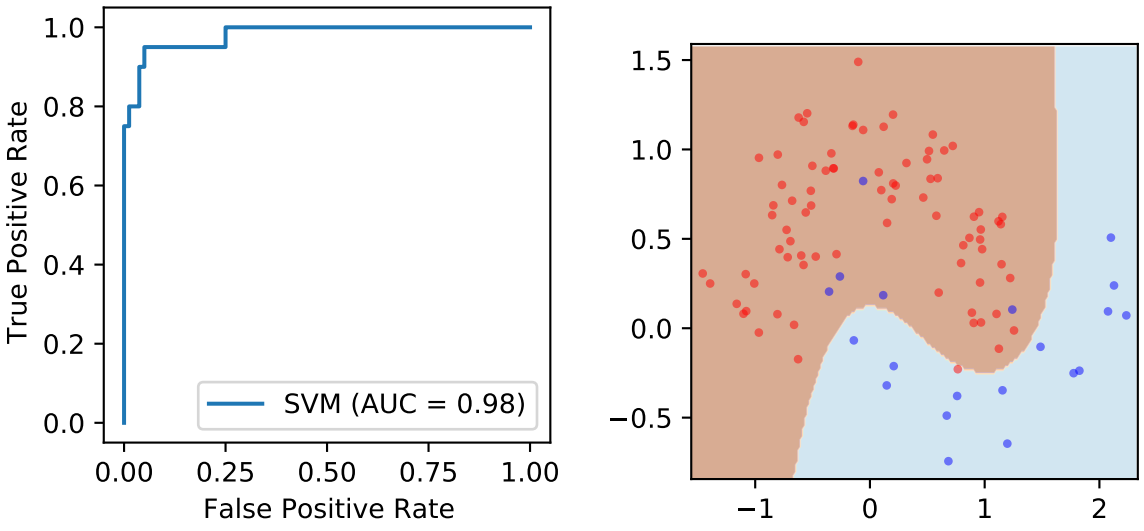

Figure 1.19: Example of *ROC* curve (left) on a toy example (right). The classifier is obtained by using a SVM. The corresponding *AUC* is 0.98.

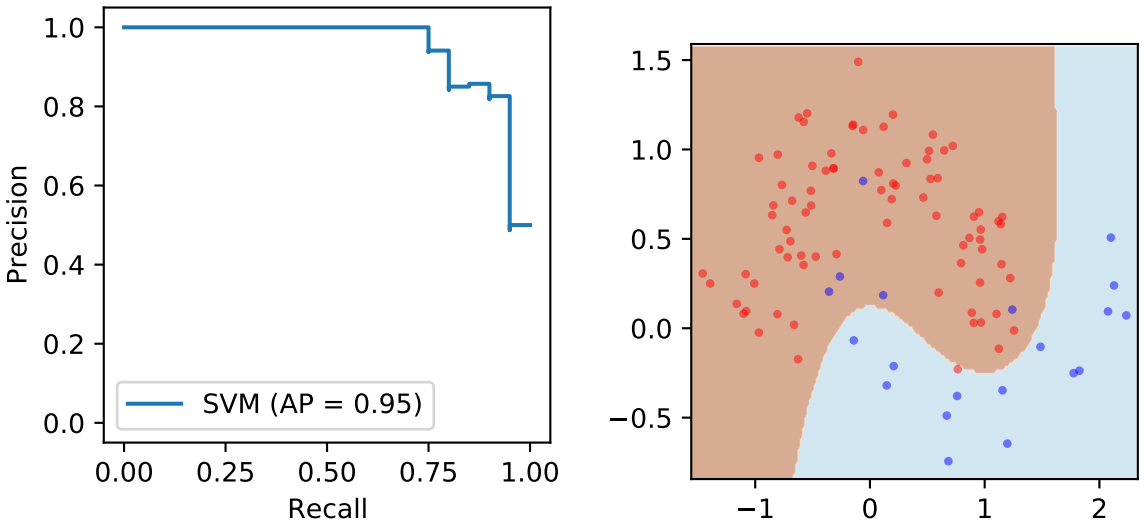

Figure 1.20: Example of *Precision-Recall* curve (left) on a toy example (right). The classifier is obtained by using a SVM. The correponding *AP* is 0.95.

Figure 1.21: Comparaison of the *AUC-ROC* and *AP* on two rankings. Blue (*resp.* grey) lines represent positive (*resp.* negative) samples. While *AUC-ROC* behaves similarly on both cases ($AUC\text{-}ROC = \frac{1}{2}$), the *Average Precision* is equal to 0.43 on the left and 0.68 on the right, illustrating that *AP* favors the ranking that put (at least some) positives at the very top of the list.

Average Precision A second performance measure that can be used in ranking is the *Average Precision* (*AP*). *AP* is the equivalent of the *AUC-ROC* but for the *Precision-Recall* curve (Figure 1.20).

Indeed, *AP* is defined as the area under this curve. It can also be defined as:

$$AP = \frac{1}{m^+} \sum_{i=1}^{m^+} p(k_i)$$

with $p(k_i)$ the *Precision* at the rank k_i, corresponding to the i^{th} positive in the ranking.

Another way to calculate it, especially when the ranking contains ties, is to use *Precision* and *Recall* directly. The formula then becomes:

$$AP = \sum_t (r_t - r_{t-1})\, p_t.$$

with r_t and p_t the *Recall* and *Precision*, respectively, at the t^th threshold of distinct prediction values.

As illustrated in Figure 1.21, unlike *AUC-ROC*, *AP* has a greater impact when one wants to focus on the top of a ranking, especially when one works under constraints, such as a limited number of possible controls.

1.5.2 Sampling Strategies

The second option to tackle the problem of learning from imbalanced datasets is to use sampling strategies. In this case, the idea is to modify the dataset to be able to use the classical algorithms optimizing a surrogate version of the accuracy.

For this, three strategies are typically used. The first one is to reduce the majority class by undersampling. The second one corresponds to the opposite case where we will increase the number of examples of the minority class. In this case, we talk about oversampling. The third one consists of a mix of the two previous by doing both under- and oversampling on the dataset. In each case, the aim is to obtain a final dataset which is almost balanced or with an acceptable imbalance rate.

Each of these methods has its pros and cons and their use will depend mainly on the data and the task at hand. Subsampling will give more weight to the minority class and allow to speed up the learning phase by reducing the size of the data set. But this will come at the expense of a loss of information because of the deletion of potentially meaningful examples. On the other hand, oversampling will give more weight to the minority class but without loosing

any information. However, this will be at the cost of a reduction of the variance (diversity) of the minority class and an increase in the size of the dataset which will have consequences on the training time. Below, we present these two strategies and some representative algorithms introduced in the literature.

Undersampling Undersampling corresponds to the removal of examples from the majority class in order to get as close as possible to the number of examples in the minority class. To this end, rather than deleting data randomly, it is preferable to work with the specificities of the dataset.

The first category of methods includes algorithms that select the irrelevant examples to be removed from the majority class. Among the main existing methods, one can cite the *Edited Nearest Neighbors* (*ENN*) algorithm [124] or *Tomek's links* [108].

In the former, the idea is to remove from the dataset the negative examples that do not agree with their neighborhood. One looks at the k nearest neighbors of a query point and classifies it by a majority vote. All misclassified negative points are then removed. Once the dataset is 'cleaned', it is possible to repeat the process in an iterative way as proposed by [1]. In this case, we can either continue with the same number of neighbors as the previous iteration (*Repeated ENN*), or increase it in order to be less drastic in terms of reduction (*Allk*−NN).

In [108], *Tomek's link* is built between two points if they are each the nearest neighbor of the other. In other words, there is a *Tomek's link* between $\mathbf{x}_1$ and $\mathbf{x}_2$ if and only if:

$$\forall \mathbf{x} \in S, \mathcal{D}(\mathbf{x}_1, \mathbf{x}_2) < \mathcal{D}(\mathbf{x}_1, \mathbf{x}) \text{ and } \mathcal{D}(\mathbf{x}_1, \mathbf{x}_2) < \mathcal{D}(\mathbf{x}_2, \mathbf{x}).$$

Once the *Tomek's links* have been found in the dataset, the points of the majority class which are linked to a point of the minority class are then removed from the dataset.

As illustrated in Figure 1.22, we can see that both methods focus mainly on the overlapping areas of the classes to remove examples from the majority class. Since these methods only work on these areas, it is possible that the dataset is still imbalanced after applying the algorithm. Their interest is more to be combined with oversampling algorithms to achieve class balance, as we can see in the paragraph 1.5.2.

The second category consists of methods that select the relevant negative examples to be kept in the majority class. Among them, one a cite the *Condensed Nearest Neighbor* algorithm [55] which aims at keeping a minimal subset able to correctly predict all the original examples by a 1−NN. Thus, one iteratively adds to this subset all the training points that would not be correctly predicted by the current subset. It starts with a subset containing the entire minority class to which one iteratively adds relevant negative examples until stabilization.

The *Near-Miss* method [79] is another strategy which offers three alternatives. The examples of the majority class are here selected according to a criterion based on their distance from the minority class. In *Near-Miss*-1, the selected examples will be those with the smallest average distance to their three nearest negative neighbors. In *Near-Miss*-2, they will be those with the smallest average distance to their three most distant neighbors. Finally, in *Near-Miss*-3, only the nearest neighbour of each example in the minority class will be kept.

As illustrated in Figure 1.23, these methods remove a lot of negative data outside the overlapping area.

The third family of strategies corresponds to methods composed of a mix of the two previous ideas. In *One Sided Selection* [66], *Tomek's links* are first used for the deletion of ambiguous points then a *Condensed Nearest Neighbor* is performed to reduce the number of negative examples in 'safe' zones, *i.e.* noisy and far from the boundaries.

In the *Neighborhood Cleaning Rule* proposed by [68], *ENN* is first used to identify the noisy data in the majority class. Then, the *Condensed Nearest Neighbor* is applied to select

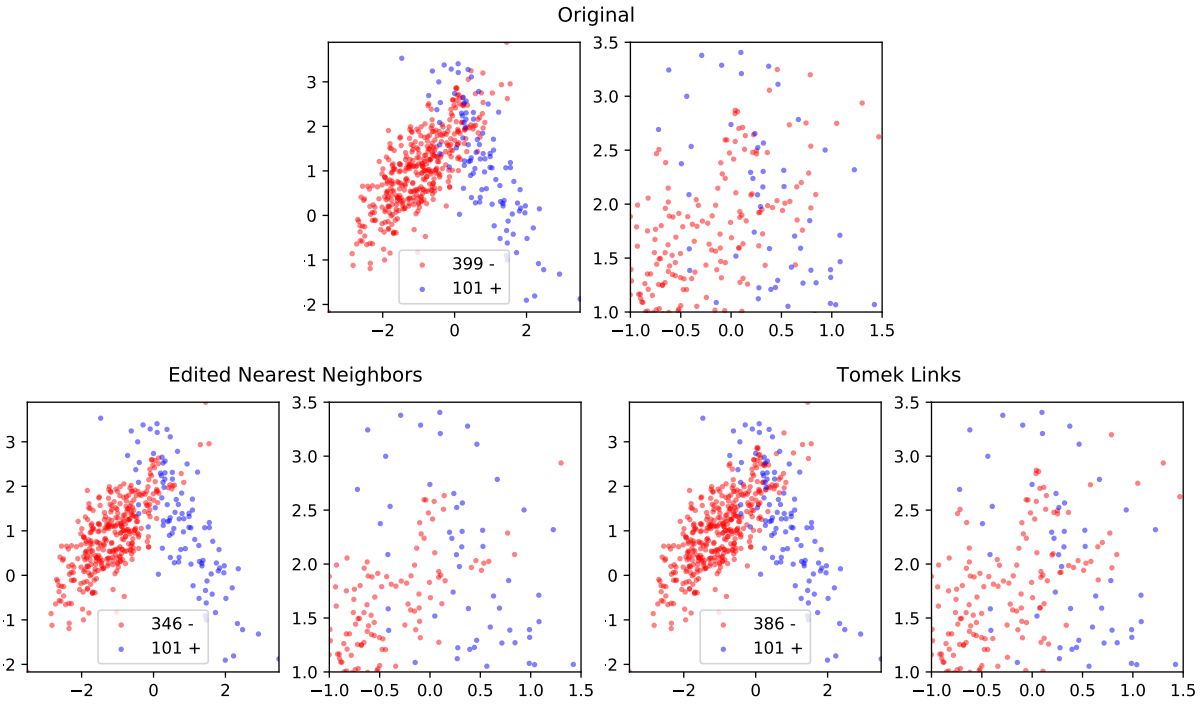

Figure 1.22: Illustrations of ENN and $Tomek's\ links$ algorithms for undersampling from a toy dataset. (Top) The original dataset. (Bottom Left) The dataset after undersampling with ENN algorithm. (Bottom Right) The dataset after undersampling with $Tomek's\ links$ algorithm. For each algorithm, (Left) the complete dataset after undersampling, with the number of examples in each class, (Right) a zoom on the intersection area of the two classes.

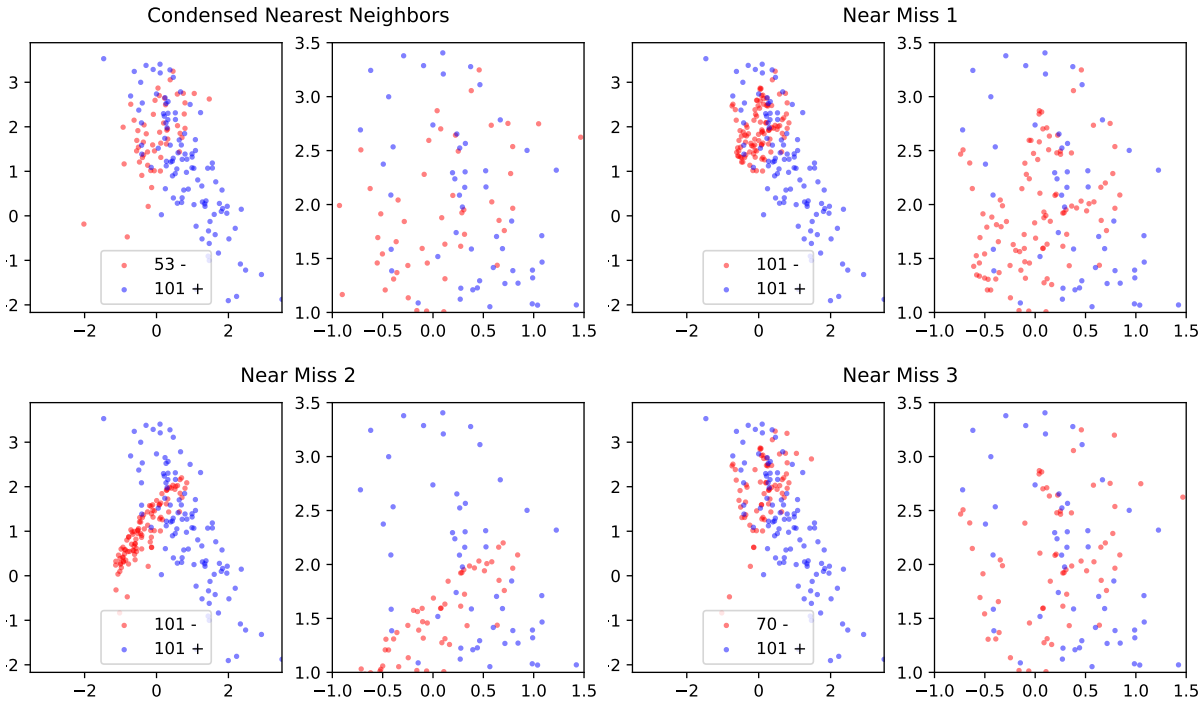

Figure 1.23: Illustrations of the *Condensed Nearest Neighbors* and *Near-Miss* algorithms for undersampling on the same toy dataset as Figure 1.22. (Top Left quarter) The dataset after undersampling with *Condensed Nearest Neighbors*. (Top Right quarter) The dataset after undersampling with the first version of the *Near-Miss* algorithm. (Bottom Left quarter) The dataset after undersampling with *Near-Miss*-2. (Bottom Right quarter) The dataset after undersampling with *Near-Miss*-3. For each algorithm, (Left) the complete dataset after undersampling, with the number of examples in each class, (Right) a zoom on the intersection area of the two classes.

a part of the remaining negative examples.

As we can see in Figure 1.24, these methods work on both areas, the overlapping one and the safe one, without being as drastic as the previous technics on this latter part of the space.

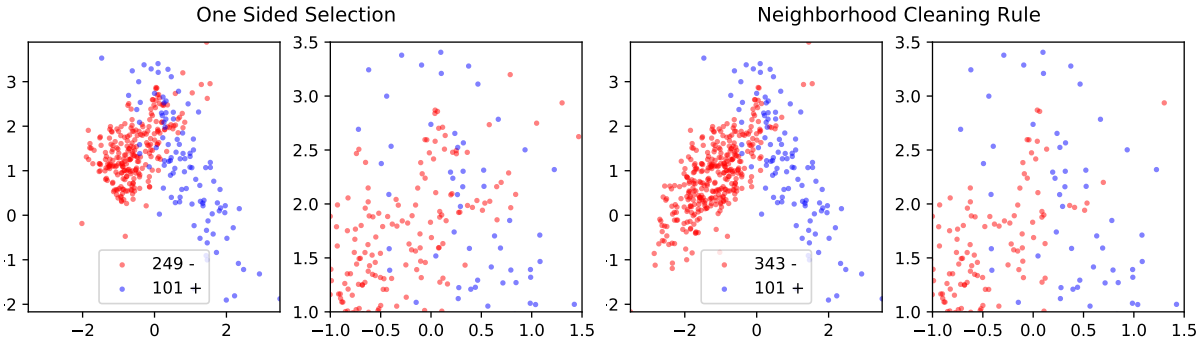

Figure 1.24: Illustrations of the *One Sided Selection* method and the *Neighborhood Cleaning Rule* for undersampling on the toy dataset of Figure 1.22. (Left) The dataset after undersampling with *One Sided Selection*. (Right) The dataset after undersampling with *Neighborhood Cleaning Rule*. For each algorithm, (Left) the complete dataset after undersampling, with the number of example in each class, (Right) a zoom on the intersection area of the two classes.

Oversampling Instead of trying to reduce the size of the set of majority examples, oversampling-based methods aim at increasing the size of the minority class. This can naively be done in a totally random way by duplicating the existing points, thus giving them more weight. However, this strategy does not bring enough diversity and therefore does not usually lead to good results. This is why synthetic example generation methods have been proposed, like $SMOTE$ [23] and its variants [54, 86] or $ADASYN$ [56].

With $SMOTE$, for $Synthetic\ Minority\ Over-sampling\ TEchnique$, artificial positive points are generated between two points of the minority class. For that, for each positive $\mathbf{x}$, one randomly chooses another positive $\mathbf{x}'$ in its k nearest neighbors and generates a synthetic example randomly on the segment $[\mathbf{x}, \mathbf{x}']$. The process is repeated until the desired IR is reached.

Its main variants are *Borderline-SMOTE* [54] and *Bordeline-SMOTE SVM* [86]. In *BorderLine-SMOTE*, the idea is to focus only on the decision boundary between classes to generate the synthetic examples. To do this, only samples from the minority class with a majority of neighbors in the other class are used for the synthetic point generation. This subset is noted 'danger'. The others are set aside as being safe enough. In a first version, the synthetic examples are generated between the points of the 'danger' set and their nearest neighbors of the minority class. In a second version, the synthetic examples are also generated from the 'danger' set, but this time considering all the nearest neighbors, whatever their class. In *Borderline-SMOTE SVM*, as its name indicates, an SVM is used to select the points, instead of a $k-$NN. Samples of the minority class that are close to the support vectors are used for the generation of synthetic points.

In $ADASYN$, for *ADAptive SYNthetic sampling* approach, the idea is to generate examples of the minority class according to their local distribution in the feature space. In the areas where they are poorly represented compared to the majority class, $SMOTE$ is used to generate a lot of synthetic examples. Only a few are generated in the areas where the density of positives is more important. This method again allows us to focus on ambiguous areas, which are more difficult to label.

In Figure 1.25, we can see that *SMOTE* generates examples in each zone containing minority examples, in a more or less homogeneous way to redensify them, whereas its variants focus more on the overlapping zones.

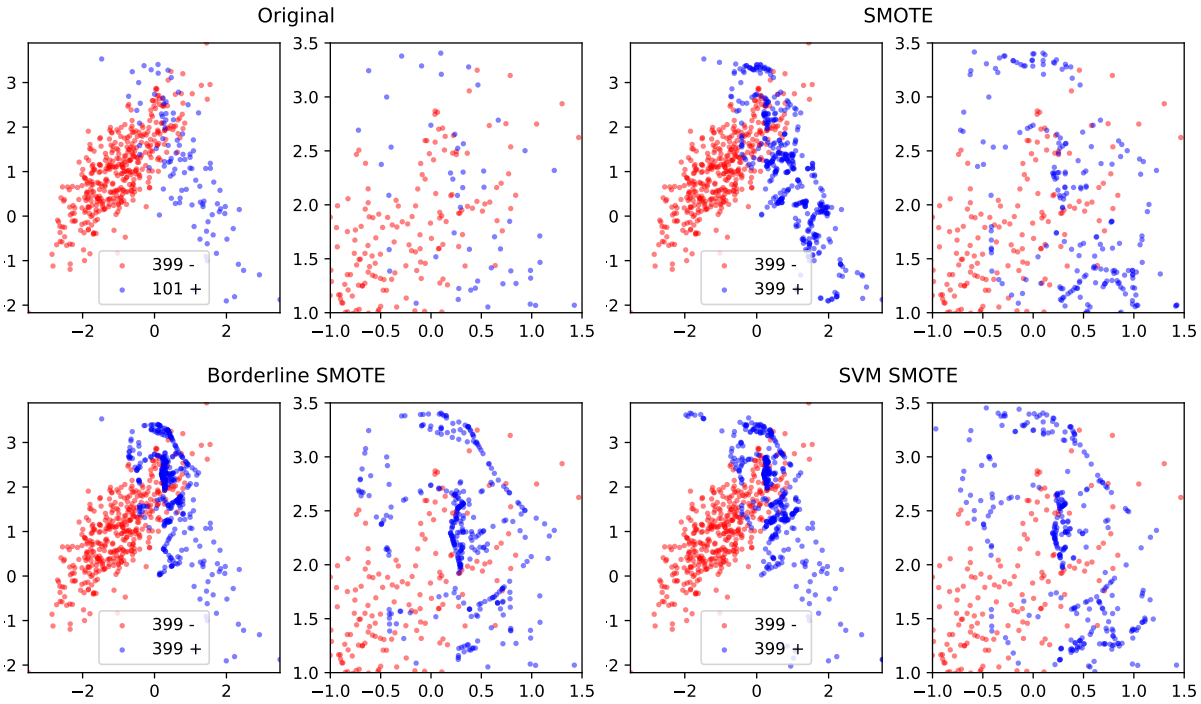

Figure 1.25: Illustrations of *SMOTE*, *Borderline-SMOTE* and *Borderline-SMOTE SVM* for oversampling on a toy dataset. (Top Left quarter) The Original dataset. (Top Right quarter) The dataset after oversampling with *SMOTE*. (Bottom Left quarter) The dataset after oversampling with *Borderline-SMOTE*. (Bottom Right quarter) The dataset after oversampling with *Borderline-SMOTE SVM*. For each algorithm, (Left) the complete dataset after oversampling, with the number of examples in each class, (Right) a zoom on the intersection area of the two classes.

Combination of undersampling and oversampling To overcome the limits of each of the sampling methods, it is quite common to combine them in order to benefit from their complementary advantages. The main idea always remains the same whatever the combination. It starts by generating the synthetic data and then cleans the dataset obtained with one of the undersampling algorithms. All combinations are clearly possible. As an illustration, we only present two of them in Figure 1.26, the combination of *SMOTE* with *ENN* or with the *Tomek's links*, available in the package [71].

In the *SMOTE+ENN* version, we see that the *ENN* algorithm cleans up the overlapping area by removing points from both classes, either real or dummy points for the minority class, to clear the decision boundary and increases the margin between the two classes. The same idea holds for the *SMOTE+Tomek* version but less drastically, which allows to keep a little more information on this contentious area.

Another strategy to address the imbalance of the dataset consists in introducing cost coefficients in the learning algorithms and so increasing the influence of the minority interest class. The underlying idea is presented in the next section.

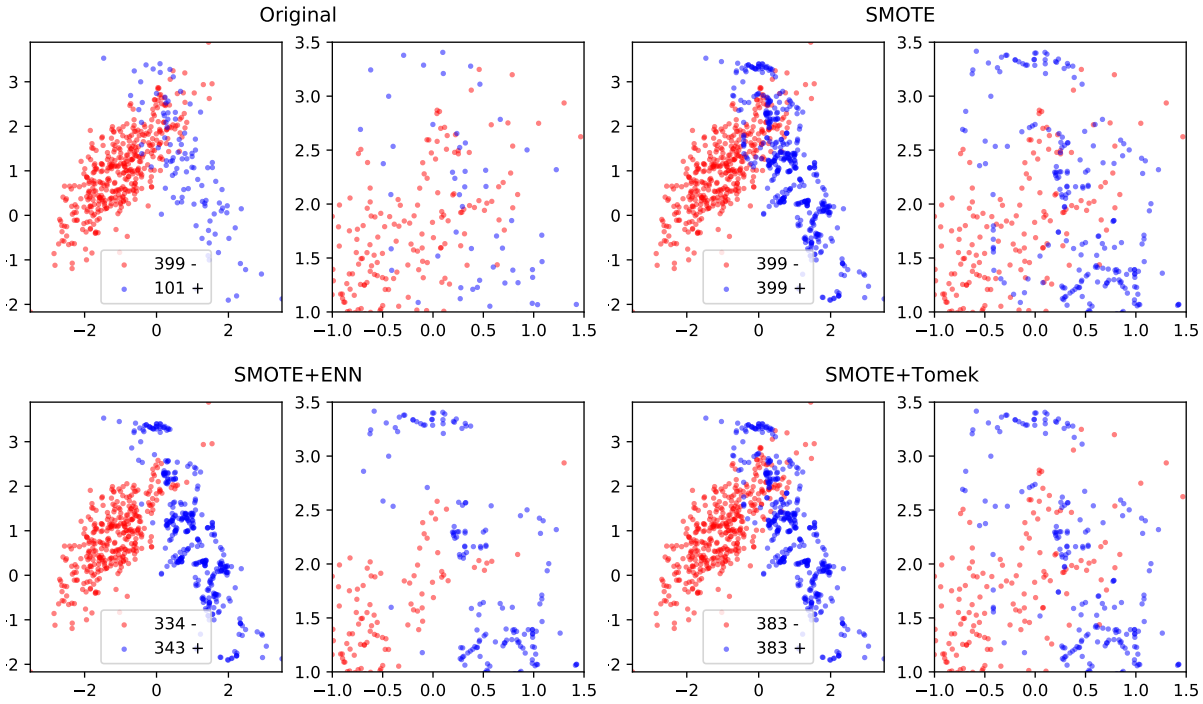

Figure 1.26: Illustrations of *SMOTE+ENN* and *SMOTE+Tomek* for sampling on a toy dataset. (Top Left quarter) The Original dataset. (Top Right quarter) The dataset after oversampling with *SMOTE*. (Bottom Left quarter) The dataset after sampling with *SMOTE* and *ENN*. (Bottom Right quarter) The dataset after sampling with *SMOTE* and *Tomek's links*. For each algorithm, (Left) the complete dataset after sampling, with the number of examples in each class, (Right) a zoom on the intersection area of the two classes.

1.5.3 Cost-Sensitive Learning

In the imbalanced context of fraud detection, it can be more damaging to miss a fraudster than to check someone who is not. Thus, the errors made by the model will not have the same impact for the user and a false negative will often lead to a larger loss than a false positive. The algorithms we have seen above generally assume that each type of error is equivalent. At the DGFiP, considering that missing a fraud might cost a large amount of money, it appears to be interesting to weight the two errors (FP, FN) differently in order to make the algorithms sensitive to the loss suffered by the algorithm for the application at hand. This is the goal of Cost-Sensitive Learning [36]. The algorithms developed in this sub-field of Machine Learning can therefore naturally be used in the case of imbalanced classification, even though this domain of Machine Learning covers a broader spectrum of applications [109].

In Cost-Sensitive Learning for imbalanced classification, instead of assuming that the errors in the confusion matrix have the same value, they will be weighted differently in order to adapt to the task. The idea is to attempt to reduce one of the types of error, even if it means slightly increasing the other, which is supposed to be less damaging. Thus, to each type of error, or even to each error, we will associate a cost C, as we can see in the following confusion matrix in Table 1.2.

Table 1.2: Confusion Matrix in a binary classification setting. C_{TP}, C_{FN}, C_{FP} and C_{TN} are the different costs for TP, FN, FP and TN.

		Predicted labels	
		$+1$	-1
True labels	$+1$	C_{TP}, TP	C_{FN}, FN
	-1	C_{FP}, FP	C_{TN}, TN

Using this confusion matrix in a learning algorithm boils down to generating a model that minimizes the global cost. For instance, if we choose to set a cost of 0 for the good predictions TP and TN, a cost of 3 for the FP and a cost of 100 for the FN, to penalise the latter, we will then have a total cost to be minimised defined by:

$$Total\ Cost = C_{FN} * FN + C_{FP} * FP = 100 * FN + 3 * FP.$$

These costs depend on the problem at hand and should therefore be carefully chosen. In practice, setting them is a tricky task. For the DGFiP, for example, it is quite simple to define the cost of a FP by evaluating the money needed for processing the case by an agent. On the other hand, the evaluation of the cost of a FN is a complex multidimensional function. Often, a first starting point is to choose costs that are inversely proportional to the rates of positives and negatives in the training set. Thus, in the example above, we might start from the fact that there are 3 positives for 100 negatives in the data set.

Once these basics are established, we can now study the different Cost-Sensitive Learning methods that can be used for classification in an imbalanced context. These often correspond to modifications of classical algorithms to make them Cost-Sensitive.

SVM [128] can be used in this context by playing with on the parameter C. Indeed, C is used to find the trade-off between the size of the margin and the amount of violations. With a global parameter and the over-representation of the majority class, the latter would be largely favoured, which does not suit us in the imbalanced context. To prevent this, it is possible to choose a value per class for C, or even per example, and integrate it directly into the problem formulation. Thus, by giving a high value for the examples of the minority class and a low value for the majority samples, we will then allow the algorithm to be more flexible on the poorly represented examples, thus tolerating a little more violations on this class, and harder for the others, thus avoiding badly classified examples by penalizing them more.

The SVM soft margin primal formulation becomes then:

$$\begin{aligned}&\min_{\mathbf{w},b,\boldsymbol{\xi}} \frac{1}{2}||\mathbf{w}||_2^2 + \sum_{i=1}^{m} C_i\xi_i \\ &\text{subject to } y_i(\langle \mathbf{w}, x_i\rangle + b) \geq 1 - \xi_i, \text{ for } 1 \leq i \leq m \\ &\text{and } \boldsymbol{\xi} \succeq 0\end{aligned}$$

with $\boldsymbol{\xi}$, the slack variables to allow some instances to violate the margin constraint but with a penalization if it happens, and $\mathbf{C} \geq 0$ the tradeoff parameters between the size of the margin and the number of violations. In practice, the C_i's are tuned by cross-validation.

Cost-Sensitive Learning can also be done directly with Logistic Regression, for which we can also define a cost per class and integrate it directly into the definition of the optimization problem which then becomes:

$$\max_{\theta} \sum_{i} \left(w_0 \; y_i \log\left(h_\theta\left(\mathbf{x}_i\right)\right) + w_1 \left(1 - y_i\right) \log\left(1 - h_\theta\left(\mathbf{x}_i\right)\right)\right)$$

with w_0 the cost associated with the misclassified examples of the majority class and w_1 the cost associated with the minority class. By taking a low w_0 and a high w_1, inversely proportional to the rates of each class, we improve the performance measure (like F_1-*score*) of the Logistic Regression in the imbalanced context. Indeed, in this case, the majority class will be considered less important and the coefficients will have a lower update. On the contrary, with a high coefficient, the minority class will have more weight in the update of the coefficients and will therefore be better captured by the model.

Finally, we can also mention the case of the Decision Trees for which a Cost-Sensitive version has also been proposed [107]. By weighting the probabilities during the calculation of the Gini index or the Entropy, one can take into account the class imbalance and improve the splitting criterion so that it does not neglect the minority class. The probability p_i to be in the class i, with $i \in \{+, -\}$ in the binary case, which is originally defined by $\frac{m_i}{m}$ thus becomes:

$$p_i = \frac{w_i m_i}{\sum\limits_j w_j m_j}$$

with $w_i = C_i \frac{m}{\sum\limits_j C_j m_j}$ and C_i the cost of misclassifying a class i instance. As in the previous examples, by giving a cost inversely proportional to the rate of positives and negatives, the imbalance is better taken into account and the minority class plays a larger role in the splitting decision.

Note that it is possible to adapt ensemble algorithms by adding this Cost-Sensitive aspect in order to make them even more powerful in the imbalanced context. This is for example the case with AdaCost [38] or AdaC1, AdaC2 and AdaC3 [104] which are Cost-Sensitive adaptations of AdaBoost.

In addition to these Cost-Sensitive algorithms, there are more specific algorithms for learning in a highly imbalanced context. These algorithms are often combinations of several techniques that have been proven to work in the imbalanced context separately, and which, when combined, give even better results. These techniques are briefly presented in the next section with a focus on two of them.

1.5.4 Specific Ensemble Algorithms

The survey [49] proposes a taxonomy of ensemble methods in the imbalanced context, thus offering an overview of some possible combinations and adaptations of ensemble algorithms. In

Algorithm 1.3: Easy Ensemble Algorithm

Input: a learning sample $S = \mathcal{P} \cup \mathcal{N}$ with $m^+ = |\mathcal{P}|$ minority examples and $m^- = |\mathcal{N}|$ majority examples, a number of subsets T and $\{s_i\}_{i=1}^T$ the number of iterations to train the AdaBoost Ensemble h_i.

Output: a final classifier $\mathcal{H}$

for *i from 1 to T* **do**

Randomly sample a subset $\mathcal{N}_i$ from $\mathcal{N}$ such that $|\mathcal{N}_i| = m^+$

Learn an AdaBoost classifier h_i using $\mathcal{P} \cup \mathcal{N}_i$, with s_i weak classifiers $h_{i,j}$ and corresponding weights $\alpha_{i,j}$ and threshold θ_i

$$h_i(\mathbf{x}) = sign\left(\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(\mathbf{x}) - \theta_i\right)$$

end

return The final classifier

$$\mathcal{H}(\mathbf{x}) = sign\left(\sum_{i=1}^{T}\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(\mathbf{x}) - \sum_{i=1}^{T} \theta_i\right)$$

particular, it includes the Cost-Sensitive adaptations of Boosting mentioned above.

Given that preprocessing by sampling methods provides improved results in the imbalanced context and that ensemble methods are already performing well, trying combinations of the two appeared quite natural and intuitive.

Thus, methods based on Boosting [24, 100] or Bagging [120, 106, 58] combined with sampling have been widely studied. For combinations with Bagging, the idea is generally to add sampling to the Boostrap in order to rebalance the classes and thus take better advantage of the strength of the ensemble of classifiers. For the combinations with Boosting, the sampling is applied in the learning loop to rebalance the classes in order not to neglect the minority class.

In the following, we briefly present Easy Ensemble and Balance Cascade Classifiers [76].

Easy Ensemble As already mentioned, one of the major problems with undersampling is the loss of information about the majority class. To overcome this limit, the Easy Ensemble algorithm trains several classifiers h_i on balanced subsets of the training set. These subsets are composed of all the minority examples and random samples of the majority examples with replacement. This ensures that all the information contained in the examples of the majority class is taken into account. Each h_i is the output of AdaBoost and the final classifier is a combination of these h_i using the weights learned in each Booster, as shown in Algorithm 1.3. Thus, Easy Ensemble can be seen as a set of ensembles, which will benefit from both the strength of Bagging and Boosting.

Balance Cascade While Easy Ensemble takes the information from the majority class in a random way, Balance Cascade does it in a supervised way. Once the first classifier h_1 has been learned, the algorithm looks for the correctly predicted examples of the majority class and removes them from the training set, assuming that the information they provide is somehow redundant with what it already knows. Doing this at each iteration, the training set contains fewer and fewer examples of the majority class and is eventually balanced. Indeed, since at

Algorithm 1.4: Balance Cascade Algorithm

Input: a learning sample $S = \mathcal{P} \cup \mathcal{N}$ with $m^+ = |\mathcal{P}|$ minority examples and $m^- = |\mathcal{N}|$ majority examples, a number of subsets T and $\{s_i\}_{i=1}^{T}$ the number of iterations to train the AdaBoost Ensemble h_i.

Output: a final classifier $\mathcal{H}$

$f \rightarrow \sqrt[T-1]{m^+/m^-}$ *// Initialization of the False Positive rate that h_i should achieve*

for *i from 1 to T* **do**

Randomly sample a subset $\mathcal{N}_i$ from $\mathcal{N}$ such that $|\mathcal{N}_i| = m^+$

Learn an AdaBoost classifier h_i using $\mathcal{P} \cup \mathcal{N}_i$, with s_i weak classifiers $h_{i,j}$ and corresponding weights $\alpha_{i,j}$ and threshold θ_i

$$h_i(\mathbf{x}) = sign\left(\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(\mathbf{x}) - \theta_i\right)$$

Adjust θ_i such that h_i's FP rate is f

Remove from $\mathcal{N}$ all examples correctly classified by h_i

end

return The final classifier

$$\mathcal{H}(\mathbf{x}) = sign\left(\sum_{i=1}^{T}\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(\mathbf{x}) - \sum_{i=1}^{T} \theta_i\right)$$

each iteration we keep $f \times m^-$ examples of the majority class (the FP examples), after $T-1$ iterations, there will remain $f^{T-1}m^- = m^+$ examples of this class. Apart from this particular sampling phase, the rest of the algorithm is identical to Easy Ensemble, as can be seen in Algorithm 1.4.

1.6 Conclusion

In this chapter, we have presented the basics of Machine Learning. We started by introducing some generalities about the principle of Empirical Risk Minimisation with respect to some loss function. Then we briefly presented the methodology of Supervised Learning, before describing some popular algorithms, notably the $k-$NN and Decision Trees which will have an important role in the rest of this thesis. Given the importance of distance functions in most of the learning algorithms, we have also presented the field of Metric Learning. Finally, we have pointed out the specificities of learning in an imbalanced context, highlighting the methods and adaptations necessary to solve this complex task.

The aim of this thesis is to develop new methods for Imbalance Learning, in particular that of fraud detection. In the following three chapters, we present our contributions to this field.

Chapter 2 presents an adaptation of the nearest neighbour algorithm so that positive examples are more important in the decision. By weighting the distance to the examples of interest, we simulate that these latter are moving closer to the test points, thus allowing them to be better considered. The intuition behind this first contribution is that fraudsters often try to mimic the behavior of non-fraudsters and therefore have a borderline attitude. Thus, by arti-

ficially moving the positive examples closer to the query, we modify the decision boundaries of the classifier.

Chapter 3 goes a step further by optimizing a specific Mahalanobis distance that induces a distortion of the feature space. This chapter generalizes the first contribution allowing one to have different weights depending on the dimensions of the space. This formalism prompted us to derive theoretical guarantees thanks to the Uniform Stability framework.

Finally, in order to fulfill the last expectation of the DGFiP on interpretability, Chapter 4 presents a model based on meta-Decision Trees. The *Average Precision* is optimized at all stages of the construction. This measure has been shown to be the most adapted criterion to the imbalanced context with restricted budget. By reading the leaves of the final tree from left to right, we obtain a ranking of potential fraudsters that is simple and easily understandable for end-users.

Chapter 2

An Adjusted Nearest Neighbor Algorithm for Imbalanced Classification

This chapter is based on the following publications

Rémi Viola, Rémi Emonet, Amaury Habrard, Guillaume Metzler, Sébastien Riou and Marc Sebban. A Nearest Neighbor Algorithm for Imbalanced Classification. In *International Journal on Artificial Intelligence Tools*, doi:10.1142/S0218213021500135. 2021 [117]. [1]

Rémi Viola, Rémi Emonet, Amaury Habrard, Guillaume Metzler, Sébastien Riou and Marc Sebban. An Adjusted Nearest Neighbor Algorithm Maximizing the F-Measure from Imbalanced Data. In *IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 2019, Portland, United States [113].

Rémi Viola, Rémi Emonet, Amaury Habrard, Guillaume Metzler, Sébastien Riou and Marc Sebban. Une Version Corrigée de l'Algorithme des Plus Proches Voisins pour l'Optimisation de la F-Mesure dans un Contexte Déséquilibré. In *Conférence sur l'Apprentissage automatique (CAp)*, 2019, Toulouse, France [114].

Abstract

In this chapter, we tackle the challenging problem of imbalanced learning from a Nearest-Neighbor (NN) classification perspective. Assuming that at the decision boundaries the positive examples look like the majority samples, we introduce an algorithm based on a simple geometrical idea of rescaling the distance between a query sample and any positive training examples. This leads to a modification of the Voronoi regions and thus of the decision boundaries of the NN classifier. As the *Accuracy* cannot be used to measure the effectiveness of our method to address this challenging problem, we use several alternative measures proposed in the literature and introduced in the previous Chapter. We provide a theoretical justification about this scaling scheme which inherently aims at reducing the False Negative rate while controlling the number of False Positives. We further formally establish a link between the proposed method and cost-sensitive learning. An extensive experimental study is conducted on many public imbalanced datasets, but also on large-scale private datasets from the DGFiP on the tax fraud detection task, showing that our method is very effective with respect to popular NN algorithms, comparable to state-of-the-art sampling methods and even yields the best performance when combined with them.

[1] This article is an extended version of the paper presented at ICTAI'2019 and selected among the best paper to be published in the IJAIT Journal.

2.1 Introduction

While the machine learning community can benefit nowadays from larger and larger datasets for optimizing provably accurate classifiers, we have seen before that in many real world applications the positive examples are very scarce compared with the number of negative samples [3, 22, 8]. This is typically the case in intrusion detection, health care insurance or bank fraud identification, and more generally anomaly detection, *e.g.*, in medicine or in industrial processes. In such a setting, the training set is composed of a few positive examples (*e.g.*, the frauds) and a huge amount of negative samples (*e.g.*, the genuine transactions). As explained in the previous Chapter, standard learning algorithms struggle to deal with this imbalance scenario because they are typically based on the minimization of (a surrogate of) the 0-1 loss. Therefore, a trivial solution consists in assigning the majority label to any test query, leading to a high performance from an accuracy perspective but completely missing the (positive) examples of interest. To overcome this issue, we have seen that several strategies have been developed over the years. The first one consists in the optimization of loss functions based on measures that are more appropriate for this context such as the *Area Under the ROC Curve* (AUC), the *Average Precision* (AP), the *G-mean* (GM), the *Balanced Accuracy* (BA) or the *F-measure* to cite a few [17, 41, 103]. The main pitfalls related to such a strategy concern the difficulty to directly optimize non smooth, non separable and non convex measures (see [7] for the specific case of the *F-measure*). A simple and usual solution to fix this problem consists in using off-the-shelf learning algorithms (maximizing the accuracy) and a posteriori pick the model with the highest AP, GM, BA or *F-measure*. Unfortunately, this might be often suboptimal. A more elaborate solution aims at designing differentiable versions of the previous non-smooth measures and optimizing them, *e.g.*, as done by gradient boosting in [43] with a smooth surrogate of the AP. The second family of methods is based on the modification of the distribution of the training data using sampling strategies [40], as presented in Section 1.5.2. This is typically achieved by removing examples from the majority class, as done, *e.g.*, in *Edited Nearest Neighbors*(ENN) [124] or *Tomek's Link* [108], and/or by adding examples from the minority class, as in *SMOTE* [23] and its variants, or by resorting to generative adversarial models [53]. One peculiarity of imbalanced learning can be interpreted from a geometric perspective. As illustrated in Figure 2.1 (left) which shows the Voronoi cells on an artificial imbalanced dataset (where two adjacent cells have been merged if they concern examples of the same class), the regions of influence of the positive examples are much smaller than that of the negatives. This explains why at test time, in imbalanced learning, the risk to get a False Negative (*e.g.*, a fraud that is wrongly classified as a genuine transaction) is high. A large number of False Negatives (FN) leads to a dramatic decrease of the aforementioned measures that all rely on a fine balance between FN and the number of False Positives FP. Note that increasing the regions of influence of the positives would mechanically reduce FN. However, not controlling the expansion of these regions, as illustrated in Figure 2.1 (right), may have a dramatic impact on FP, and thus on the previous performance measures.

The main contribution of this chapter is about the problem of finding the appropriate trade-off (Figure 2.1 (middle)) between the two above-mentioned extreme situations (large FP or FN, both leading to a poor performance at test time). A natural way to increase the influence of positives may consist in using generative models (like GANs [53]) to sample new artificial examples, mimicking the negative training samples. However, beyond the issues related to the parameter tuning, the computation burden and the complexity of such a method, using GANs to optimize the *Precision* and *Recall* is still an open problem (see [97] for a recent study on this topic). We show in this chapter that a much simpler strategy can be used by modifying the distance exploited in a $k-$Nearest Neighbor ($\boldsymbol{k-}$**NN**) algorithm [29] which enjoys many interesting advantages, including its simplicity, its capacity to approximate asymptotically any locally regular density, and its theoretical rootedness [78, 65, 64]. $\boldsymbol{k-}$**NN** also benefited from

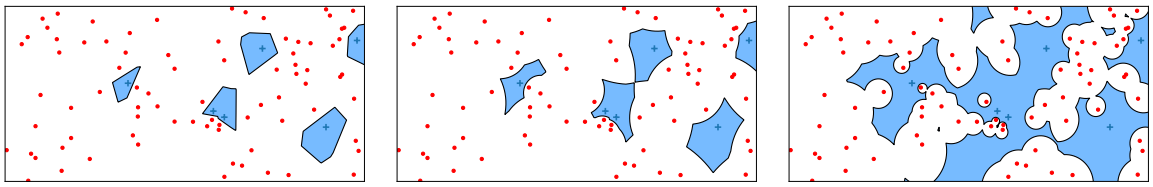

Figure 2.1: Toy imbalanced dataset: On the left, the Voronoi regions around the positives are small. The risk to generate False Negatives (FN) at test time is large. On the right: by increasing too much the regions of influence of the positives, the probability to get False Positives (FP) grows. In the middle: an appropriate trade-off between the two previous situations.

many algorithmic advances during the past decade in the field of metric learning, aiming at optimizing under constraints the parameters of a metric, typically the Mahalanobis distance, as done in *LMNN* [123] or *ITML* [30] (see [10, 9] for a survey and Section 1.4 of this manuscript). Unfortunately, existing metric learning methods are dedicated to enhance the $\boldsymbol{k}$**−NN** accuracy and do not focus on the optimization of criteria, like the *F-measure* or *G-mean*, in scenarios where the positive training examples are scarce. A geometric solution to increase, at a very low cost, the region of influence of the minority class consists in modifying the distance when comparing a query example to a positive training sample. More specifically, we formally show in this chapter that the optimization of any (FN,FP)-based performance measure, which are well suited to deal with imbalanced scenarios, is facilitated by scaling the distance to any positive by a coefficient $\gamma \in [0, 1]$ leading to the expansion of the Voronoi cells around the minority examples. An illustration is given in Figure 2.1 (middle) which might be seen as a good compromise that results in the reduction of FN while controlling the risk to increase FP. Note that our strategy boils down to modifying the local density of the positive examples. For this reason, we claim that it can be efficiently combined with *SMOTE*-based sampling methods whose goal is complementary and consists in generating examples on the path linking two (potentially far) positive neighbors. Our experiments will confirm this intuition.

The rest of the chapter is organized as follows. Section 2.2 briefly reminds some notations and performance measures already introduced in the previous Chapter. The related work is presented in Section 2.3. Section 2.4 is devoted to the presentation of our method $\boldsymbol{\gamma k}$**−NN**. This section includes a theoretical analysis of our method as well as a presentation of a local extension aiming at capturing local specificities of the feature space. We finally establish a link between $\boldsymbol{\gamma k}$**−NN** and cost-sensitive learning. The last part of this chapter is dedicated to an extensive experimental study performed on 28 imbalanced public datasets and 11 private datasets from the French Ministry of Economy and Finance on the tax fraud detection task (see Section 2.5). In this comparative analysis, we give evidence of the complementarity of our method with sampling strategies. We also present a qualitative analysis on the *MNIST* dataset which allows a visualisation of the impact of our algorithm. We finally conclude in Section 2.6.

2.2 Notations and Evaluation Measures

We consider a training sample $S = \{(\mathbf{x}_i, y_i), i = 1, ..., m\}$ of size m, drawn from an unknown joint distribution $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X} = \mathbb{R}^d$ is the feature space and $\mathcal{Y} = \{-1, +1\}$ is the set of labels. Let us assume that $S = S_+ \cup S_-$ with m_+ positives $\in S_+$, m_- negatives $\in S_-$ and $m = m_+ + m_-$.

Learning from imbalanced datasets requires to optimize appropriate measures that take into account the scarcity of positive examples. As seen in the previous Chapter, several of them rely on the following two quantities: the Recall (also called *True Positive Rate* (TPR) or *sensitivity*)

which measures the capacity of the model to recall/detect positive examples, and the Precision (also called *Positive Prediction Value* (PPV)) which is the confidence in the prediction of a positive label. For the sake of completeness, we recall their definitions here:

$$\text{Recall} = \frac{TP}{TP + FN} \quad \text{and} \quad \text{Precision} = \frac{TP}{TP + FP},$$

where FP (resp. FN) is the number of false positives (resp. negatives) and TP is the number of true positives. Since one can arbitrarily improve the Precision if there is no constraint on the Recall (and vice-versa), they are usually combined into a single measure.
For instance, the *F-measure* [92] (or F_β score), which is widely used in fraud and anomaly detection [52], is defined as the harmonic mean of the Recall and Precision:

$$F_\beta = (1 + \beta^2)\frac{\text{Precision} \times \text{Recall}}{\beta^2 \times \text{Precision} + \text{Recall}},$$

where β is set such that the Recall is considered β times as important as the Precision. Note that F_1 (*i.e.* $\beta = 1$) considers the Precision and Recall equally.
The *G-measure* (G_1) can also be used for imbalanced data classification [37]. Unlike F_1, it rather considers the geometric mean of Precision and Recall:

$$G_1 = \sqrt{\text{Precision} \times \text{Recall}}.$$

While F_β and G_1 consider both Recall and Precision, the *G-mean* (GM) [66] rather makes use of the Recall (or TPR) and the True Negative Rate (TNR) as follows:

$$GM = \sqrt{TPR \times TNR} = \sqrt{\frac{TP}{TP + FN} \times \frac{TN}{TN + FP}}.$$

In other words, it computes the geometric mean of TPR and TNR. Thus, it gives a higher importance to the negative class, compared to the previous measures. Without being exhaustive, a last performance measure that can be used in an imbalanced setting is the *Balanced Accuracy* (BA) [17] which also relies on TPR and TNR and is defined as the mean accuracy of the two classes:

$$BA = (TPR + TNR)/2$$

In the experimental section of this chapter, we will resort to these widely used measures to assess the efficiency of our proposed method to overcome the problem of scarcity of positive samples.

2.3 Related Work

In this section, we present the main strategies that have been proposed in the literature to address the problem of learning from imbalanced datasets by resorting to a $\boldsymbol{k}-\mathbf{NN}$ classifier.

The oldest method is certainly the one presented in [34] which consists in associating to each neighbor a voting weight that is inversely proportional to its distance to a query point $\mathbf{x}$. The assigned label $\hat{y}$ of $\mathbf{x}$ is defined as:

$$\hat{y} = \left| \sum_{\mathbf{x}_i \in kNN(\mathbf{x})} y_i \times \frac{1}{\mathcal{D}(\mathbf{x}, \mathbf{x}_i)} \right|,$$

where $kNN(\mathbf{x})$ stands for the set of the k nearest neighbors of $\mathbf{x}$. A more refined version consists in taking into account both the distances to the nearest neighbors and the distribution of the features according to the class $p(\mathbf{x}_i \mid y_i)$ [75]. Despite these modifications in the decision rule,

the sparsity of the positives remains problematic and it is possible that no positives fall in the neighborhood of a new query $\mathbf{x}$. To tackle this issue, a variant of $\boldsymbol{k}-\mathbf{NN}$, called kPNN [133], is to consider the region of the space around a new query $\mathbf{x}$ which contains exactly k positive examples. By doing so, the authors are able to use the density of the positives to estimate the probability of belonging to the minority class.

A more recent version has been shown to perform better than the two previous approaches: $\boldsymbol{k}$**RNN** [134]. If the idea remains similar (*i.e.* estimating the local sparsity of minority examples around a new query), the posterior probability of belonging to the minority class is adjusted so that it considers both the local and global disequilibrium for the estimation. In [6], the authors use both the label and the distance to the neighbors $(\mathbf{x}_i, y_i)$ to define a scaled metric $\mathcal{D}'$ from the Euclidean distance $\mathcal{D}$, as follows:

$$\mathcal{D}'(\mathbf{x}, \mathbf{x}_i) = \left(\frac{m_i}{m}\right)^{1/d} \mathcal{D}(\mathbf{x}, \mathbf{x}_i),$$

where m_i is the number of examples in the class y_i. As we will see later, this method falls in the same family of strategies as our contribution, aiming at scaling the distance to the examples according to their label. However, three main differences explain the superiority of our method, observed in the experiments: (i) $\boldsymbol{k}$**RNN** fixes $\mathcal{D}'$ in advance while we will automatically adapt the scaling factor to optimize the considered performance measure; (ii) because of (i), $\mathcal{D}'$ needs to take into account the dimension d of the feature space (and so will tend to $\mathcal{D}$ as d grows) while our method captures the intrinsic dimension of the space by selecting the best weight; (iii) $\mathcal{D}'$ is useless when combined with sampling strategies (indeed, $\frac{m_i}{m}$ would tend to be uniform) while our method will allow us to scale differently the distance to the original positive examples and the ones artificially generated.

Another way to assign weights to each class, which is close to the sampling methods, is to duplicate the positive examples according to the Imbalance Ratio $IR = m_-/m_+$. Thus, it can be seen as a *uniform* over-sampling technique, where all positives are replicated the same number of times. However, note that this method requires to work with $k > 1$.

A last family of methods that try to improve $\boldsymbol{k}-\mathbf{NN}$ is related to *metric learning* [9, 10]. *LMNN* [123] or *ITML* [30] are two famous examples which optimize under constraints a Mahalanobis distance $\mathcal{D}_{\mathbf{M}}(\mathbf{x}, \mathbf{x}_i) = \sqrt{(\mathbf{x} - \mathbf{x}_i)^\top \mathbf{M} (\mathbf{x} - \mathbf{x}_i)}$ parameterized by a positive semi-definite (PSD) matrix $\mathbf{M} \in \mathbb{S}^+$. As presented in Section 1.4, such methods seek a linear projection of the data in a latent space where the Euclidean distance is applied. As we will see in the following, our scaling method is a special case of metric learning which looks for a diagonal matrix (but applied only when comparing a query to a positive example) and which behaves well whatever the considered performance measure.

Interestingly, we can note that all the previous sampling methods try to overcome the problem of learning from imbalanced data by resorting to the notion of k-neighborhood. This is justified by the fact that $\boldsymbol{k}-\mathbf{NN}$ has been shown to be a good estimate of the density at a given point in the feature space.

In our contribution, we also leverage $\boldsymbol{k}-\mathbf{NN}$ but with a different approach. Instead of generating (many) new examples (which would have a negative impact from a complexity perspective), we locally modify the density around the positive points. We achieve this by rescaling the distance between a test sample and the positive training examples. We show that such a strategy can be efficiently combined with sampling methods, whose goal is complementary, by potentially generating new examples in regions of the space where the minority class is not present.

2.4 Proposed Approach

In this section, we present our $\boldsymbol{\gamma k}-$**NN** method which works by scaling the distance between a query point and positive training examples by a factor.

2.4.1 An Adjusted $k-$NN algorithm

Statistically, when learning from imbalanced data, a new query $\mathbf{x}$ has more chance to be close to a negative example due to the rarity of positives in the training set, even around the mode of the positive distribution. We have seen two families of approaches that can be used to counteract this effect: (i) creating new synthetic positive examples, and (ii) changing the distance according to the class. The approach we propose falls into the second category.

We suggest to modify how the distance to the positive examples is computed, in order to compensate for the imbalance in the dataset. We artificially bring a new query $\mathbf{x}$ closer to any positive data point $\mathbf{x}_i \in S_+$ in order to increase the effective area of influence of positive examples. The new measure $\mathcal{D}_\gamma$ that we propose is defined using an underlying distance $\mathcal{D}$ (*e.g.*, the Euclidean distance) as follows:

$$\mathcal{D}_\gamma(\mathbf{x}, \mathbf{x}_i) = \begin{cases} \mathcal{D}(\mathbf{x}, \mathbf{x}_i) & \text{if } \mathbf{x}_i \in S_-, \\ \gamma \cdot \mathcal{D}(\mathbf{x}, \mathbf{x}_i) & \text{if } \mathbf{x}_i \in S_+. \end{cases} \tag{2.1}$$

As we will tune the γ parameter, this new way to compute the similarity to a positive example is close to a Mahalanobis-distance learning algorithm, looking for a PSD matrix, as previously described. However, the matrix $\mathbf{M}$ is restricted here to be $\gamma^2 \cdot \mathbf{I}$, where $\mathbf{I}$ refers to the identity matrix. Moreover, while metric learning typically works by optimizing a convex loss function under constraints, our γ is simply tuned such as maximizing some non convex performance measure. Lastly, and most importantly, it is applied only when comparing the query to positive examples. As such, $\mathcal{D}_\gamma$ is not a proper distance. However, this is what allows it to compensate for the class imbalance. In the binary setting, there is no need to have a γ parameter for the negative class, since only the relative distances are used. In the multi-class setting with K classes, we would have to tune up to $K-1$ values of γ.

Before formalizing the $\boldsymbol{\gamma k}-$**NN** algorithm that will leverage the distance $\mathcal{D}_\gamma$, we illustrate in Figure 2.2, on 2D data, the decision boundary induced by a nearest neighbor binary classifier that uses $\mathcal{D}_\gamma$. We consider an elementary dataset with only two points, one positive and one negative. The case of $\gamma = 1$, which is a traditional $1-$NN is shown in a thick black line. Lowering the value of γ below 1 brings the decision boundary closer to the negative point, and eventually tends to surround it very closely. Fig 2.3 shows, with more complex (toy) datasets, that γ controls how much we want to push the boundary towards negative examples. Fig 2.3 (right) should be imagined as a zoomed-in boundary between the classes, where one class is 20 times less represented. It shows that, due to sampling, the $1-$NN boundary wrongly causes regions of false negatives, while $\boldsymbol{\gamma k}-$**NN** is able to correct the bias.

We can now present $\boldsymbol{\gamma k}-$**NN** (see Algorithm 2.1) that is parameterized by the γ parameter. It has the same overall complexity as $\boldsymbol{k}-$**NN**. The first step to classify a query $\mathbf{x}$ is to find its k nearest negative neighbors and its k nearest positive neighbors. Then, the distances to the positive neighbors are multiplied by γ, to obtain $\mathcal{D}_\gamma$. These $2k$ neighbors are then ranked and the k closest ones are used for classification (with a majority vote, as in $\boldsymbol{k}-$**NN**). It should be noted that, although $\mathcal{D}_\gamma$ does not define a proper distance, we can still use any existing fast nearest neighbor search algorithm, because the actual search is done only using the original distance $\mathcal{D}$ (but twice, once for S_+, once for S_-).

2.4.2 Theoretical analysis

In this section, we formally analyze what could be a good range of values for γ in our $\boldsymbol{\gamma k}-$**NN** algorithm. To this aim, we study what impact γ has on the probability to get a False Positive

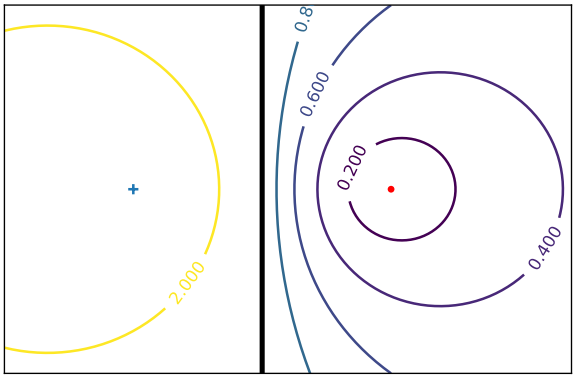

Figure 2.2: Evolution of the decision boundary based on $\mathcal{D}_\gamma$, for a 1−NN classifier, on a 2D dataset with one positive (resp. negative) instance represented by a blue cross (resp. red point). The value of γ is given on each boundary ($\gamma = 1$ on the thick line).

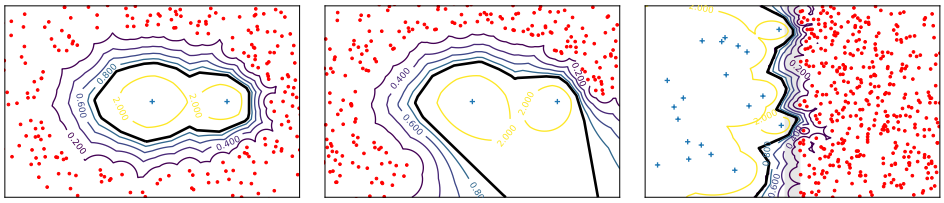

Figure 2.3: Behavior of the decision boundary according to the γ value for the 1−NN classifier on toy datasets. Positive points are shown as blue crosses and negatives ones as red dots. The black line represents the standard decision boundary for the 1−NN classifier, *i.e.* when $\gamma = 1$.

(and False Negative) at test time and explain why it is important to choose $\gamma < 1$ when the imbalance in the data is significant. The following analysis is made for $k = 1$ but note that the conclusion still holds for $k > 1$.

Proposition 1. *(False Negative probability) Let* $\mathcal{D}_\gamma(\mathbf{x}, \mathbf{x}_+) = \gamma \cdot \mathcal{D}(\mathbf{x}, \mathbf{x}_+)$, $\forall \gamma > 0$, *be our modified distance used between a query* $\mathbf{x}$ *and any positive training example* $\mathbf{x}_+$, *where* $\mathcal{D}(\mathbf{x}, \mathbf{x}_+)$ *is some distance function. Let* $FN_\gamma(\mathbf{z})$ *be the probability for a positive example* $\mathbf{z}$ *to be a False Negative using Algorithm (2.1). The following result holds: if* $\gamma \leq 1$,

$$FN_\gamma(\mathbf{z}) \quad \leq \quad FN(\mathbf{z})$$

Proof. (sketch of proof) Let ϵ be the distance from $\mathbf{z}$ to its nearest-neighbor $N_\mathbf{z}$. $\mathbf{z}$ is a false negative if $N_\mathbf{z} \in S_-$ that is all positives $\mathbf{x}' \in S_+$ are outside the sphere $\mathcal{S}_{\frac{\epsilon}{\gamma}}(\mathbf{z})$ centered at $\mathbf{z}$ of

Algorithm 2.1: Classification of a new example with $\boldsymbol{\gamma k}$**−NN**.

Input : a query $\mathbf{x}$ to be classified, a set of labeled samples $S = S_+ \cup S_-$, a number of neighbors k, a positive real value γ, a distance function $\mathcal{D}$

Output: the predicted label of $\mathbf{x}$

$\mathcal{NN}^-, \mathfrak{D}^- \leftarrow nn(k, \mathbf{x}, S_-)$ *// nearest negative neighbors with their distances*

$\mathcal{NN}^+, \mathfrak{D}^+ \leftarrow nn(k, \mathbf{x}, S_+)$ *// nearest positive neighbors with their distances*

$\mathfrak{D}^+ \leftarrow \gamma \cdot \mathfrak{D}^+$

$\mathcal{NN}_\gamma \leftarrow firstK\left(k, sortedMerge((\mathcal{NN}^-, \mathfrak{D}^-), (\mathcal{NN}^+, \mathfrak{D}^+))\right)$

$y \leftarrow \oplus$ if $\left|\mathcal{NN}_\gamma \cap \mathcal{NN}^+\right| \geq \frac{k}{2}$ else $\ominus$ *// majority vote based on* $\mathcal{NN}_\gamma$

return y

radius $\frac{\epsilon}{\gamma}$. Therefore,

$$\begin{aligned} FN_\gamma(\mathbf{z}) &= \prod_{\mathbf{x}' \in S_+} \left(1 - P(\mathbf{x}' \in \mathcal{S}_{\frac{\epsilon}{\gamma}}(\mathbf{z}))\right), \\ &= \left(1 - P(\mathbf{x}' \in \mathcal{S}_{\frac{\epsilon}{\gamma}}(\mathbf{z}))\right)^{m_+} \end{aligned} \tag{2.2}$$

while

$$FN(\mathbf{z}) = \left(1 - P(\mathbf{x}' \in \mathcal{S}_\epsilon(\mathbf{z}))\right)^{m_+}. \tag{2.3}$$

Solving (2.2) $\leq$ (2.3) implies $\gamma \leq 1$. □

This result means that satisfying $\gamma < 1$ allows us to increase the decision boundary around positive examples (as illustrated in Figure 2.3), yielding a smaller risk to get False Negatives at test time. An interesting comment can be made from Eq.(2.2) and (2.3) about their convergence. As m_+ is supposed to be very small in imbalanced datasets, the convergence of $FN(\mathbf{z})$ towards 0 is pretty slow, while one can speed-up this convergence with $FN_\gamma(\mathbf{z})$ by increasing the radius of the sphere $\mathcal{S}_{\frac{\epsilon}{\gamma}}(\mathbf{z})$, that is taking a small value for γ.

Proposition 2. *(False Positive probability) Let $FP_\gamma(\mathbf{z})$ be the probability for a negative example $\mathbf{z}$ to be a False Positive using Algorithm (2.1). The following result holds: if $\gamma \geq 1$,*

$$FP_\gamma(\mathbf{z}) \leq FP(\mathbf{z})$$

Proof. (sketch of proof) Using the same idea as before, we get:

$$\begin{aligned} FP_\gamma(\mathbf{z}) &= \prod_{\mathbf{x}' \in S_-} \left(1 - P(\mathbf{x}' \in \mathcal{S}_{\gamma\epsilon}(\mathbf{z}))\right), \\ &= \left(1 - P(\mathbf{x}' \in \mathcal{S}_{\gamma\epsilon}(\mathbf{z}))\right)^{m_-} \end{aligned} \tag{2.4}$$

while

$$FP(\mathbf{z}) = \left(1 - P(\mathbf{x}' \in \mathcal{S}_\epsilon(\mathbf{z}))\right)^{m_-}. \tag{2.5}$$

Solving (2.4) $\leq$ (2.5) implies $\gamma \geq 1$. □

As expected, this result suggests to take $\gamma > 1$ to increase the distance $\mathcal{D}_\gamma(\mathbf{z}, \mathbf{x}_+)$ from a negative test sample $\mathbf{z}$ to any positive training example $\mathbf{x}_+$ and thus reduce the risk to get a False Positive. It is worth noticing that while the two conclusions from Propositions 1 and 2 are contradictory, the convergence of $FP_\gamma(\mathbf{z})$ towards 0 is much faster than that of $FN_\gamma(\mathbf{z})$ because $m_- >> m_+$ in an imbalanced scenario. Therefore, fulfilling the requirement $\gamma > 1$ is much less important than satisfying $\gamma < 1$. For this reason, we will impose our Algorithm (2.1) to take $\gamma \in [0, 1]$. As we will see in the experimental section, the more imbalance the datasets, the smaller the optimal γ, confirming the previous conclusion.

2.4.3 Link with cost-sensitive learning

In this section, we show that it is possible to establish a link between the cost-sensitive learning framework [36] and our algorithm $\boldsymbol{\gamma k}-\mathbf{NN}$. As presented in Section 1.5.3, the goal of cost-sensitive learning is to assign different costs to each entry of the confusion matrix as depicted in Table 2.1 for a binary setting where we will denote the 4 costs as c_{TP}, c_{FN}, c_{FP} and c_{TN}. Cost sensitive methods are widely used, including in imbalanced scenarios, to give more importance (*i.e.* higher weights/costs) to the examples of the positive/minority class. By doing so, a learned classifier will focus more on decreasing the loss associated to the positive samples. We show

Table 2.1: Cost matrix for a binary classification task.

	Predicted positive	Predicted negative
Actual positive	c_{TP}	c_{FN}
Actual negative	c_{FP}	c_{TN}

here that, despite not being learned by optimizing a loss function, $\boldsymbol{\gamma k-}$**NN** can still be seen through the lens of cost-sensitive learning.

Let us assume that the correct predictions are not penalized, *i.e.* $c_{TN} = c_{TP} = 0$ and that c_{FP} and c_{FN} are such that $c_{FP} + c_{FN} = 1$ (without loss of generality, as only their relative values matter here). Let $\mathbf{x}^-$ (resp. $\mathbf{x}^+$) be the nearest negative (resp. positive) neighbor of an example $\mathbf{x}$. Suppose that we have a model $\eta(\mathbf{x}) = \mathbb{P}(y = 1 \mid \mathbf{x})$ that gives the probability for $\mathbf{x}$ to be positive. Then the positive label will be assigned to $\mathbf{x}$ if $\eta(\mathbf{x}) > 1/2$, without considering the costs of miss-classification. Taking these latter into account changes the classification rule. Indeed, to minimize the cost-sensitive risk, an example $\mathbf{x}$ must be predicted positive if:

$$\begin{aligned} c_{FP}\,\mathbb{P}(y=0 \mid \mathbf{x}) &\leq c_{FN}\,\mathbb{P}(y=1 \mid \mathbf{x}), \\ \Leftrightarrow c_{FP}\,(1-\eta(\mathbf{x})) &\leq c_{FN}\,\eta(\mathbf{x}), \\ \Leftrightarrow \eta(\mathbf{x}) &\geq c_{FP}. \end{aligned}$$

On the other hand, our algorithm $\boldsymbol{\gamma k-}$**NN** classifies an example $\mathbf{x}$ as positive if $\mathcal{D}(\mathbf{x}, \mathbf{x}^-) > \gamma\mathcal{D}(\mathbf{x}, \mathbf{x}^+)$. Given this classification rule, we can show that $\boldsymbol{\gamma k-}$**NN** resorts to an approximation $\hat{\eta}(\mathbf{x})$ of $\eta(\mathbf{x})$ for a given weighted problem, as follows:

$$\begin{aligned} \mathcal{D}(\mathbf{x}, \mathbf{x}^-) &> \gamma\mathcal{D}(\mathbf{x}, \mathbf{x}^+), \\ \Leftrightarrow \mathcal{D}(\mathbf{x}, \mathbf{x}^-)(1+\gamma) &> \gamma(\mathcal{D}(\mathbf{x}, \mathbf{x}^+) + \mathcal{D}(\mathbf{x}, \mathbf{x}^-)), \\ \Leftrightarrow \frac{\mathcal{D}(\mathbf{x}, \mathbf{x}^-)}{\mathcal{D}(\mathbf{x}, \mathbf{x}^+) + \mathcal{D}(\mathbf{x}, \mathbf{x}^-)} &> \frac{\gamma}{1+\gamma}. \end{aligned}$$

Setting $c_{FP} = \dfrac{\gamma}{\gamma+1}$ (and therefore $c_{FN} = \dfrac{1}{\gamma+1}$) and $\hat{\eta}(\mathbf{x}) = \dfrac{\mathcal{D}(\mathbf{x}, \mathbf{x}^-)}{\mathcal{D}(\mathbf{x}, \mathbf{x}^+) + \mathcal{D}(\mathbf{x}, \mathbf{x}^-)}$ finishes to establish the link between $\boldsymbol{\gamma k-}$**NN** and cost-sensitive learning. Note that if $\gamma = 1$ then $c_{FP} = c_{FN} = \frac{1}{2}$ implying that we retrieve a standard $\boldsymbol{k-}$**NN** classifier which treats positive and negative samples equally without cost sensitivity.

The reader interested in cost-sensitive $\boldsymbol{k-}$**NN** classifiers can refer to [89, 131].

2.4.4 Towards a local approach of $\boldsymbol{\gamma k-}$NN

In what have been presented so far, we consider a single γ for the whole input space. While this has the advantage of having a single parameter to tune, it removes the ability to capture non-stationary class imbalance. Indeed, it is possible that a γ value is optimal in one part of the space but not in another.

We thus propose a non-stationary version of our algorithm, called *local-*$\boldsymbol{\gamma k-}$***NN***. Conceptually, we could have a $\gamma_{\mathbf{x}}$ for every position $\mathbf{x}$ in the space. However, such an over parameterized model would loose the simplicity of the proposed approach and increase the risk of overfitting. To deal with these two issues, we rather partition the input space into $q \in \mathbb{N}^*$ sub-spaces, $\{C_j\}_{j=1}^q$, using a clustering algorithm (*e.g.*, *k-means*). Then a value γ_j, for all $j = 1, ..., q$ is tuned according to the performance measure of interest and using only the available data in the subspace C_j. To classify a test query that falls in cluster j, we use $\boldsymbol{\gamma k-}$**NN** (with γ_j) in this cluster. We will show in the experimental Section 2.5.4 two possible variants of this local approach.

Table 2.2: Information about the studied public datasets sorted by imbalance ratio IR. The *target* column refers to the label chosen as the minority class (*i.e.* positive examples) in the dataset. The short name of each dataset is given first and will be used, for the sake of readability, in some graphs of this study. (*) The target for YEAST is *ME2 vs MIT,ME3,EXC,VAC,ERL*.

DATASETS	SIZE	DIM	TARGET	IR
BAL - BALANCE	625	4	*L*	1.2
AUT - AUTOMPG	392	7	*2,3*	1.7
ION - IONOSPHERE	351	34	*b*	1.8
PIM - PIMA	768	8	*positive*	1.9
GLA - GLASS	214	9	*1*	2.1
GER - GERMAN	1000	23	*2*	2.3
YE1 - YEAST1	1484	8	*NUC*	2.5
HAB - HABERMAN	306	3	*positive*	2.8
VE3 - VEHICLE3	846	18	*Class 3 Opel*	3.0
HAY - HAYES	132	4	*3*	3.4
SEG - SEGMENTATION	2310	19	*WINDOW*	6
AB8 - ABALONE8	4177	10	*8*	6.4
YE3 - YEAST3	1484	8	*ME3*	8.1
EC3 - ECOLI3	336	7	*imU*	8.6
PAG - PAGEBLOCKS	5472	10	*2,3,4,5*	8.8
SAT - SATIMAGE	6435	36	*4*	9.3
YEA - YEAST-0-5-6-7-9VS4	528	8	(*)	9.4
LIB - LIBRAS	360	90	*1*	14
Y17 - YEAST-1VS7	459	7	*VAC vs NUC*	14.3
ARR - ARRHYTHMIA	452	278	*6*	17
SOL - SOLAR-FLARE-M0	1389	32	*M0*	19
OIL - OIL	937	49	*minority*	22
YE4 - YEAST4	1484	8	*ME2*	28.1
WI4 - REDWINEQUALITY4	1599	11	*4*	29.2
YE5 - YEAST5	1484	8	*ME1*	32.7
YE6 - YEAST6	1484	8	*EXC*	41.4
A17 - ABALONE17	4177	10	*17*	71
A20 - ABALONE20	4177	10	*20*	159.7

2.5 Experiments

This part is devoted to an extensive experimental evaluation of $\boldsymbol{\gamma k}-\mathbf{NN}$ on public and real private datasets with comparisons to classic distance-based methods and state-of-the-art sampling strategies able to deal with imbalanced data. All the results for the public datasets are reported for nearest neighbor classification with $k = 1$ and 3 by considering the four different evaluation measures reminded in Section 2.2 (F_1, G_1, GM and BA). For the private datasets, we only report the results for $k = 3$ and the F_1 measure. We also conduct, in Section 2.5.3, a qualitative analysis on the behavior of our approach on the famous *MNIST* image dataset [69]. Finally, we conclude our experimental study by an evaluation of the performance of the local version of $\boldsymbol{\gamma k}-\mathbf{NN}$ (in Section 2.5.4).

Table 2.3: Information about the private datasets of the DGFiP, sorted by imbalance ratio IR. The short name of each dataset is given first and will be used, for the sake of readability, in some graphs of this study.

DATASETS	SIZE	DIM	IR
DGFIP 19-2	16643	265	1.9
DGFIP 9-2	440	173	3.0
DGFIP 4-2	255	82	3.8
DGFIP 8-1	1028	255	4.6
DGFIP 8-2	1031	254	4.6
DGFIP 9-1	409	171	5.1
DGFIP 4-1	240	76	5.2
DGFIP 16-1	789	162	8.7
DGFIP 16-2	786	164	9.1
DGFIP 20-3	17584	294	19.0
DGFIP 5-3	19067	318	24.9

2.5.1 Experimental setup

For these experiments, we use 28 public datasets from the well-known UCI[2] and KEEL[3] repositories. The main properties of these datasets are summarized in Table 2.2, including the imbalance ratio IR defined as: IR$= m_-/m_+$. We also use 11 real fraud detextion datasets provided by the DGFiP. These datasets correspond to data coming from tax and VAT declarations of French companies and are used for tax fraud detection purpose covering declaration of over-valued, fictitious or prohibited charges, wrong turnover reduction or particular international VAT frauds such as "VAT carousels". As explained in the introduction of this thesis, the DGFiP performs about 50,000 tax audits per year within a panel covering more than 3,000,000 companies. Being able to select the right companies to control each year is a crucial issue with a potential high societal impact. Thus, designing efficient imbalance learning methods is key. The properties of these private datasets are summarized in Table 2.3.

All the datasets are normalized using a min-max normalization such that each feature lies in the range $[-1, 1]$. We randomly draw 80%-20% splits of the data to generate the training and test sets respectively. Hyperparameters are tuned with a 10-fold cross-validation over the training set. We repeat the process over 5 runs and average the results in terms of the four performance measures. In a first series of experiments, we compare our method $\boldsymbol{\gamma k}$**−NN** to 6 other distance-based baselines:

- the classic $k-$Nearest Neighbor algorithm, $\boldsymbol{k}$**−NN**,
- the weighted version of $\boldsymbol{k}$**−NN** using the inverse distance as a weight to predict the label, **w**$\boldsymbol{k}$**−NN** [34],
- the class weighted version of $\boldsymbol{k}$**−NN**, **cw**$\boldsymbol{k}$**−NN** [6],
- the $\boldsymbol{k}$**−NN** version where each positive is duplicated according to the IR of the dataset, **dup**$\boldsymbol{k}$**−NN**,
- $\boldsymbol{k}$**RNN** [134] where the sparsity of minority examples is taken into account by modifying the way the posterior probability of belonging to the positive class is computed.
- the metric learning method ***LMNN*** [123].

[2] `https://archive.ics.uci.edu/ml/datasets.html`
[3] `https://sci2s.ugr.es/keel/datasets.php`

The hyperparameter μ of ***LMNN***, weighting the impact of impostor constraints (see [123] for more details), is tuned in the range $[0, 1]$ using a step of 0.1. Our γ parameter is tuned in the range $[0, 1]$[4] using a step of 0.1. For ***k*RNN**, we use the parameters values as described in [134].

In a second series of experiments, we compare our method to five oversampling strategies described in Section 1.5.2: *SMOTE*, *BorderlineSMOTE*, *ADASYN*, *SMOTE* with *ENN* and *SMOTE* with *Tomek's link*. The number of generated positive examples is tuned over the set of ratios $\frac{m_+}{m_-} \in \{0.1, 0.2, ..., 0.9, 1.0\}$ and such that the new ratio is greater than the original one before sampling. The other parameters of these methods are the default ones used by the package *ImbalancedLearn* [71] of *Scikit-learn*. We report the performance of the best oversampler that we denote as OS*.

In order to evaluate how both strategies are complementary, we also combine **γ*k*−NN** with oversamplers, and use the notation (OS+**γ*k*−NN**)* to indicate the best combination obtained by a 10-cross validation. In this latter scenario, we propose to learn a different γ value to be used with the synthetic positives. Indeed, some of the synthetic examples may be generated in some True Negative areas and, in this situation, it might be more appropriate to decrease their influence. The γ parameter for these examples is tuned in the range $[0, 2]$ using a step of 0.1. Note that the upper bound of the range is now set to 2. This allows **γ*k*−NN** to adapt to the different sampling strategies of the oversamplers and enables the possibility to move synthetic positive examples away from dense regions of negatives by selecting $\gamma > 1$.

2.5.2 Analysis of the results

The results on the public datasets using the six baselines are provided in Tables 2.6,2.7,2.8 and 2.9 for the four different performance measures F_1, BA, GM and G_1 respectively. These tables report the complete results when $k = 1$ (in ***k*−NN**) and provide only the mean results over the 28 datasets for $k = 3$, for the sake of concision and because the behavior for this latter value is similar. Overall, our **γ*k*−NN** approach performs much better than its competitors by achieving an improvement of 0.7 to 5 points on average, compared to the other state-of-the-art algorithms when $k = 1$. It is worth noticing that the results are even better when $k = 3$. But the certainly most striking result comes from the capacity of **γ*k*−NN** associated with the Balanced Accuracy (BA) in Table 2.7 and G-mean (GM) in Table 2.8 to address large imbalanced learning tasks. While the other methods struggle to get good results, **γ*k*−NN** with BA and GM gets the best performance 19 and 20 times respectively over the 22 largest imbalanced datasets (from Yeast1 to Abalone20). Even the metric learning algorithm *LMNN* fails to be competitive while it optimizes a representation of the data specifically dedicated to deal with nearest neighbor classification. Indeed, *LMNN* suffers from the lack of positive data to learn an efficient projection when dealing with highly imbalanced tasks. On the other hand, **γ*k*−NN** does not seem particularly sensitive to the imbalance ratio.

To see the influence of the imbalance ratio on the optimal γ-parameter, we decide to consider the Balance dataset which has the smallest imbalance ratio and we artificially increase this ratio by iteratively randomly under-sampling the minority class over the training set. We report the results on Figure 2.4. As expected, we can observe that the optimal γ value decreases when the imbalance increases. However, note that from a certain IR (around 15), γ seems to reach a plateau required to keep a satisfactory performance measure.

The second series of experiments focuses on the use of sampling strategies and the potential interest of combining **γ*k*−NN** with a synthetic generation of additional positive examples. Figure 2.5 compactly summarizes, for the four measures of interest and for both $k = 1$ (on the

[4]We experimentally noticed that using a larger range for γ leads in fact to a potential decrease of performances due to overfitting phenomena. This behavior is actually in line with the analysis provided in Section 2.4.2.

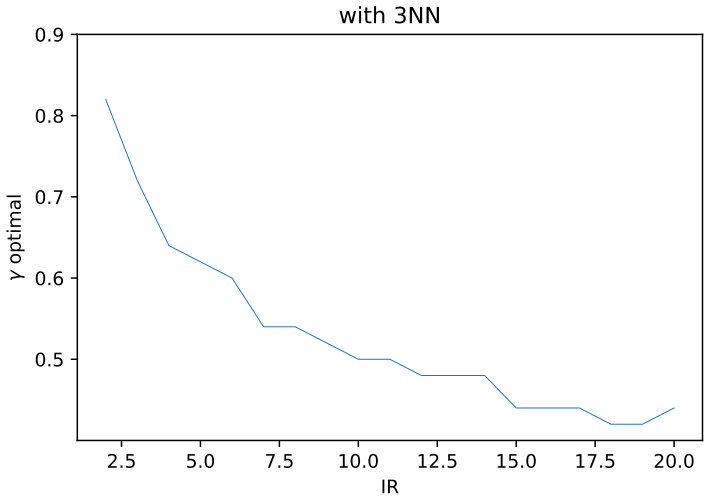

Figure 2.4: Evolution of the optimal γ value with respect to the IR on the Balance dataset, for $k = 3$.

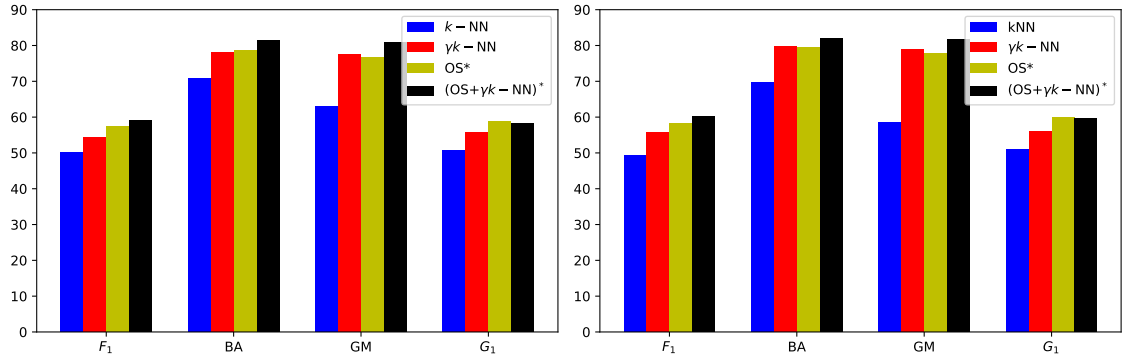

Figure 2.5: Comparison of $\boldsymbol{k}$**−NN**, $\boldsymbol{\gamma k}$**−NN**, the best oversampler among *SMOTE*, *BorderlineSMOTE*, *SMOTE+ENN*, *SMOTE+Tomek's links* and *ADASYN*, and the best coupling oversampler + $\boldsymbol{\gamma k}$**−NN**, in terms of mean of *F-measure* (F_1), *Balanced Accuracy* (BA), *Geometric Mean* (GM) and *G-measure* (G_1) over all the datasets, for $k = 1$ (left) and $k = 3$ (right).

left) and $k = 3$ (on the right), the impact of sampling strategies. Two main comments can be made from these results. First, $\boldsymbol{\gamma k}$**−NN** is complementary to the oversamplers. Indeed, for both $k = 1$ and $k = 3$ and for 3 out of 4 measures (G_1 excluded), using $\boldsymbol{\gamma k}$**−NN** in addition with a sampler leads to better results and gives evidence of the fact that $\boldsymbol{\gamma k}$**−NN** and oversamplers do not work the same way, focusing on different subparts of the feature space. While $\boldsymbol{\gamma k}$**−NN** aims at expanding the decision boundaries in favor of the positives in the neighborhood of the test query, oversamplers rather tend to fill in the empty parts of the space by generating synthetic positive examples. Second, $\boldsymbol{\gamma k}$**−NN** (red bars) alone is shown to be very competitive while benefiting from its simplicity. Indeed, we remind the reader that the performance of OS* (resp. (OS+$\boldsymbol{\gamma k}$**−NN**)*) are obtained from the costly selection of the best oversampler (resp. $\boldsymbol{\gamma k}$**−NN** + oversampler) for each dataset. Therefore, the green and black bars in Figure 2.5 give an optimistic usage of an oversampling strategy because it is generated from the average obtained over a large set of oversamplers (SMOTE, BorderlineSMOTE, ADASYN, SMOTE+ ENN and SMOTE with Tomek's link) that can be seen as an additional hyperparameter. On the other hand, in $\boldsymbol{\gamma k}$**−NN**, only one parameter (γ) is required to be tuned.

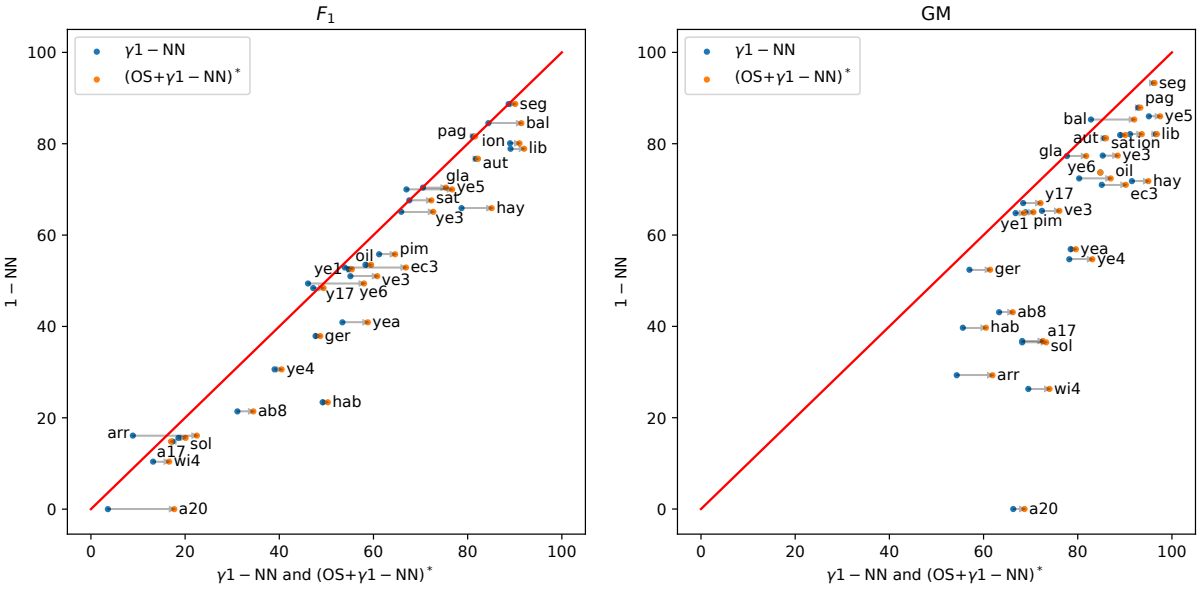

Figure 2.6: Comparison of $\boldsymbol{k}-$**NN** with (i) $\boldsymbol{\gamma k}-$**NN** (points in blue) and (ii) $\boldsymbol{\gamma k}-$**NN** coupled with the best sampling strategy (points in orange) for each dataset, in terms of *F-measure* (left) and *Geometric Mean* (right) and for $k = 1$. Points below the line $y = x$ means that $\boldsymbol{k}-$**NN** is outperformed. A move from left to right illustrates that the joint approach is better.

Figure 2.6 illustrates, for the F_1 and GM measures and $k = 1$, a dataset-wise view of the advantage of combining $\boldsymbol{\gamma k}-$**NN** with an oversampler compared to a standard $\boldsymbol{k}-$**NN**. A blue point (representing one of the 28 datasets) below the line $y = x$ means that $\boldsymbol{k}-$**NN** is outperformed. Moreover, a move of a blue point to an orange point from left to right (illustrated by a right arrow) illustrates that the joint approach leads to better results. We can see that even for the least favorable measure (*i.e.* F_1 on the left), most of the datasets are below the line and benefit from $\boldsymbol{\gamma k}-$**NN** associated to an oversampler.

In Fig 2.7 (left), we illustrate how having two γ parameters (γ on reals and γ on synthetics) gives the flexibility to independently control the influence of the actual and artificial positives respectively. The other figures (center and right) represent two examples of heatmaps of the F-measure (note that the trend is the same for the other 3 measures). We can note that while the γ parameter tuned for the real positives tends to be smaller than 1 (according to the analysis of Section 2.4.2), the γ parameter required to deal with the synthetic positives is sometimes smaller (right), sometimes greater than 1 (center), depending on the underlying density and the peculiarity of the feature space.

Recall that Propositions 1 and 2 in Section 2.4.2 tell us that selecting a γ parameter smaller than 1 for the real positives should tend to reduce the False Negative (FN) rate while still optimizing the performance measure. To illustrate our theoretical study, we plot in Figure 2.8 the percentage of FN generated by the 7 compared methods. As expected, we can note that whatever the performance measure and the value of k ($k = 1$ on the left and $k = 3$ on the right), the number of FN is much smaller than that of the competitors explaining why $\boldsymbol{\gamma k}-$**NN** gets the best results.

The results for the real datasets of the DGFiP are available in Table 2.4. Note that only the SMOTE algorithm is reported here since the other oversamplers have comparable performances. The analysis of the results leads to observations similar as the ones made for the public datasets. Our $\gamma k-$NN approach outperforms classic $k-$NN and is better than the results obtained by

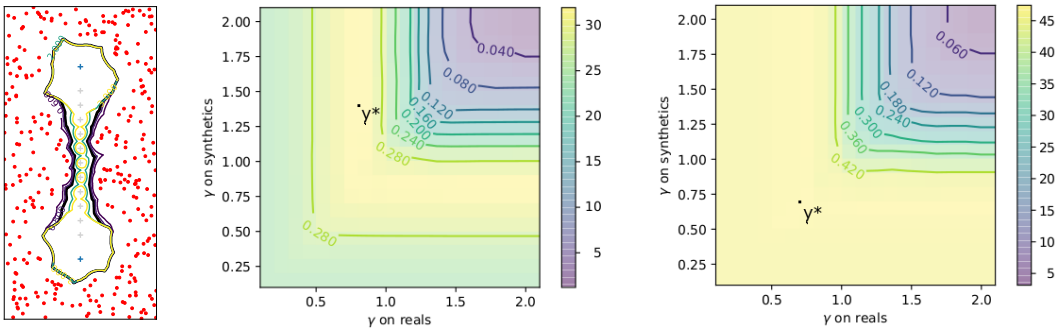

Figure 2.7: Left: Illustration on a toy dataset of the effect of varying γ for the generated positive points (in grey) while keeping a fixed $\gamma = 0.4$ for the real positives. Center and Right: Two examples of heatmap for the F-Measure that show the pair of γ (on real and synthetic positives) corresponding to the best joint approach (OS+ $\boldsymbol{\gamma k-}$**NN**)* on ABALONE8 (center) and GERMAN (right).

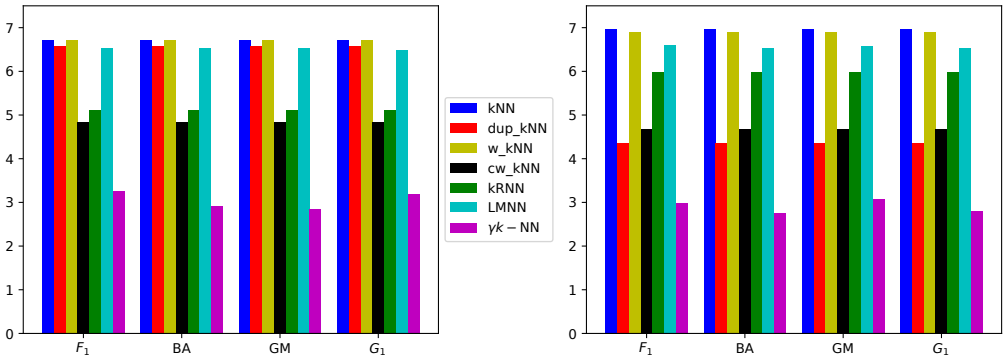

Figure 2.8: Percentage of False Negatives (FN) generated by the 7 compared methods *w.r.t.* the four performance measures with 1−NN (left) and 3−NN (right).

the SMOTE strategy. Coupling the SMOTE sampling method with our distance correction $\gamma k-$NN allows us to improve the global performance showing the applicability of our method on real data.

2.5.3 A qualitative analysis on the *MNIST* dataset

In order to visualize the qualitative impact of $\boldsymbol{\gamma k-}$**NN**, we conduct in this section some additional experiments on the *MNIST* dataset. To generate a minority class, we build 10 datasets $MNIST_i$ (one for each digit, $i = 0, ..., 9$) from the original one by considering the label i as the minority class and all the other classes representing the remaining digits as the majority class.

As previously done, a 10-fold CV is performed to find the optimal value γ. The mean results of the comparison of $\boldsymbol{\gamma k-}$**NN** with $\boldsymbol{k-}$**NN** are reported in Table 2.5 where $k = 3$. We can see that whatever the performance measure, $\boldsymbol{\gamma k-}$**NN** allows us to surpass $\boldsymbol{k-}$**NN**. As expected, the gain on this well-known *MNIST* dataset is not significant due to the already very high accuracy reached by the standard $\boldsymbol{k-}$**NN**. However, the main objective here is elsewhere. We aim at showing the quality of both the space and the neighborhoods induced by $\boldsymbol{\gamma k-}$**NN**. To illustrate this purpose and visualize how using $\mathcal{D}_\gamma$ (as defined in Eq.(2.1)) bends the feature space, we leverage t-SNE to embed the $MNIST_i$ points in 2D. Note that even if $\mathcal{D}_\gamma$ is not an actual distance (the symmetry is not satisfied), it can still be used with t-SNE that only embeds points while preserving relative pair-wise dissimilarities.

Following the definition of $\mathcal{D}_\gamma$, we scale the Euclidean distance $\mathcal{D}$ when the second point

Table 2.4: Results for 3−NN on the DGFiP datasets. The values correspond to the mean *F-measure* F_1 over 5 runs. The best result on each dataset is indicated in bold while the second is underlined.

DATASETS	3−NN	γk−NN	SMOTE	SMOTE+γk−NN
DGFiP 19-2	0.454(0.007)	<u>0.528</u>(0.005)	0.505(0.010)	**0.529**(0.003)
DGFiP 9-2	0.173(0.074)	<u>0.396</u>(0.018)	0.340(0.033)	**0.419**(0.029)
DGFiP 4-2	0.164(0.155)	<u>0.373</u>(0.018)	0.368(0.057)	**0.377**(0.018)
DGFiP 8-1	0.100(0.045)	**0.299**(0.010)	0.278(0.043)	**0.299**(0.011)
DGFiP 8-2	0.140(0.078)	0.292(0.028)	**0.313**(0.048)	<u>0.312</u>(0.021)
DGFiP 9-1	0.088(0.090)	0.258(0.036)	<u>0.270</u>(0.079)	**0.288**(0.026)
DGFiP 4-1	0.073(0.101)	<u>0.231</u>(0.139)	0.199(0.129)	**0.278**(0.067)
DGFiP 16-1	0.049(0.074)	0.166(0.065)	<u>0.180</u>(0.061)	**0.191**(0.081)
DGFiP 16-2	0.210(0.102)	0.202(0.056)	<u>0.220</u>(0.043)	**0.229**(0.026)
DGFiP 20-3	0.142(0.015)	<u>0.210</u>(0.019)	0.199(0.015)	**0.212**(0.019)
DGFiP 5-3	0.030(0.012)	0.105(0.008)	**0.110**(0.109)	<u>0.107</u>(0.010)
MEAN	0.148(0.068)	<u>0.278</u>(0.037)	0.271(0.057)	**0.295**(0.028)

Table 2.5: Comparison of $\boldsymbol{\gamma k}$**−NN** and $\boldsymbol{k}$**−NN** on *MNIST* for $k = 3$.

	F_1	BA	GM	G_1
$\boldsymbol{k}$**−NN**	0.9718	0.9831	0.9830	0.9719
$\boldsymbol{\gamma k}$**−NN**	**0.9721**	**0.9897**	**0.9897**	**0.9721**

in the pair is a positive one. Figure 2.9 compares, on the $MNIST_2$ dataset, the output of t-SNE when using $\mathcal{D}$ (left) and $\mathcal{D}_\gamma$ (right). The analysis of this embedding shows that $\mathcal{D}_\gamma$ is able to gather minority examples together in a denser cluster while the Euclidean distance $\mathcal{D}$ leads to a space where the positive samples are more scattered, some being in the middle of negative regions. This impact of $\boldsymbol{\gamma k}$**−NN** on the decision boundaries, that we see with this t-SNE experiment, is also illustrated in Figure 2.10 which shows some examples for which, the $\boldsymbol{\gamma k}$**−NN** predictions are different from that of $\boldsymbol{k}$**−NN** according to their 3 nearest neighbors (on the original dataset).

2.5.4 On local-$\boldsymbol{\gamma k}$−NN using clustering

We now evaluate our local algorithm local-$\boldsymbol{\gamma k}$**−NN** (as presented in Section 2.4.4), which partitions the input space into q clusters $(C_1, C_2, ..., C_q)$ and uses a parameter γ_j for each cluster j. The partitioning is performed using k-means, run on the training set. Note that we consider two ways of obtaining the γ_j values. The first version (V1) consists in applying the 10 fold cross-validation (CV) procedure in each cluster C_j to tune γ_j. At test time, a new point $\mathbf{x}'$ is first assigned to the nearest cluster C_k based on the closest centroid using the Euclidean distance, and the corresponding γ_k value is used to scale the distances to the positives.
We propose a second version (V2) to compensate for the fact that V1 is at risk of generating very different values of γ for two neighboring clusters. While the test decision is similar to V1, the γ_j values are obtained differently, by computing several clusterings. In V2, the 10 fold CV also includes the clustering, so 10 additional partitionings are performed. Each training point $\mathbf{x}$ will thus fall in 9 clusters (in the 9 different clusterings for which the point is not in the validation fold). Each point thus has 9 "best" γ values that we average to get a single value $\gamma_{\mathbf{x}}$ for every single point. In the end, γ_j is computed as $\frac{1}{|C_j|}\sum_{\mathbf{x}\in C_j}\gamma_{\mathbf{x}}$, *i.e.* the average γ value of the training points falling into cluster j.

The results are provided for the 4 performance measures in Figure 2.11, 2.12, 2.13 and 2.14. Despite an inherent increase of the time complexity, it is worth noting that V2 is better than the original $\boldsymbol{\gamma k}$**−NN** (on average over the 28 datasets), while V1 does not lead to improvements

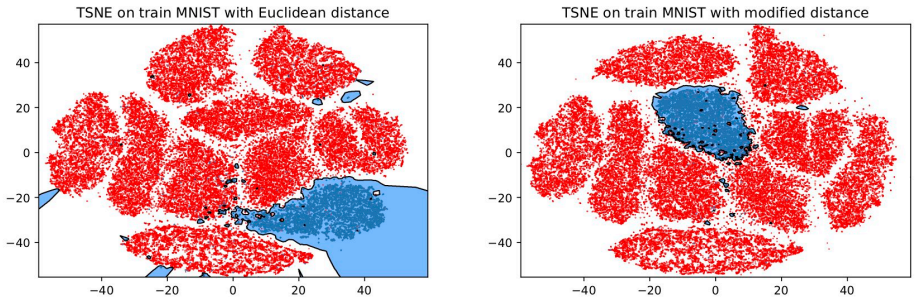

Figure 2.9: Visualization on $MNIST_2$ of the influence of the Euclidean distance $\mathcal{D}$ (left) and $\mathcal{D}_\gamma$ (right) with t-SNE. The red (resp. blue) points correspond to negatives (resp. positives). The blue areas represent the subparts of the space leading to a positive classification by a 3−NN.

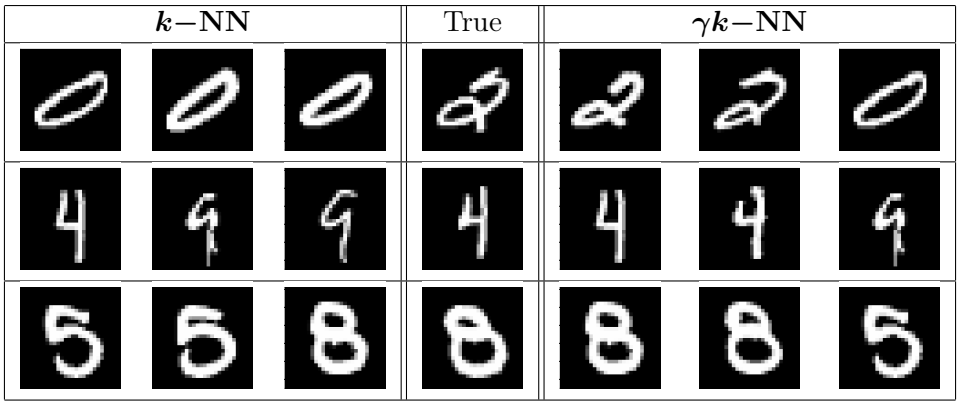

Figure 2.10: First three columns: 3 nearest neighbors using $\boldsymbol{k-}$**NN**; fourth column: the test query; last three figures: 3 nearest neighbors using $\boldsymbol{\gamma k-}$**NN**.

probably due to overfitting phenomena. Note also that in a huge majority of the datasets (around 90%), the V2 version of local-$\boldsymbol{\gamma k-}$**NN** equals or outperforms $\boldsymbol{\gamma k-}$**NN**.

2.6 Conclusion

In this chapter, we have proposed a new approach, $\boldsymbol{\gamma k-}$**NN**, that addresses the problem of learning from imbalanced datasets. It is based on the $\boldsymbol{k-}$**NN** algorithm but it modifies the distance to the positive examples by expanding the decision boundaries around these minority samples. It has been shown to outperform its competitors in terms of several performance measures. Furthermore, we gave evidence of the complementarity of $\boldsymbol{\gamma k-}$**NN** with oversampling strategies. Our algorithm, despite its simplicity, is highly effective even on real datasets and its local version local-$\boldsymbol{\gamma k-}$**NN** has shown to be even more efficient by taking the spatial specificity of the distributions into account.

This work opens the door to two promising lines of research. First, we might extend the idea of the local variant of $\boldsymbol{\gamma k-}$**NN** by proposing a multi-view learning approach, where the different results of $\boldsymbol{\gamma k-}$**NN** obtained with different subsets of features (the different views) would be combined in some way. Second, we can note that tuning γ is equivalent to building a diagonal matrix (with γ^2 in the diagonal) and applying a Mahalanobis distance only between a query and a positive example. This comment suggests to make use of metric learning to optimize a PSD matrix under (FP, FN)-based constraints that could leverage recent metric learning approaches for imbalanced data [118]. This is the goal of our second contribution presented in Chapter 3.

Table 2.6: Results for $k = 1$ with F_1 as performance measure over 5 runs. The standard deviation is indicated after the $\pm$ sign and the best results on each dataset is indicated in bold. Only the mean value when $k = 3$ is shown in the last line.

DATASETS	$k-$NN	DUP$k-$NN	W$k-$NN	CW$k-$NN	KRNN	LMNN	$\gamma k-$NN
BALANCE	0.845±0.022	0.845±0.022	0.845±0.022	0.844±0.017	**0.882±0.012**	0.841±0.046	0.844±0.017
AUTOMPG	0.767±0.074	0.767±0.074	0.767±0.074	0.762±0.062	**0.825±0.028**	0.775±0.031	0.818±0.030
IONOSPHERE	0.801±0.042	0.801±0.042	0.801±0.042	0.833±0.030	0.833±0.030	0.813±0.033	**0.890±0.045**
PIMA	0.558±0.047	0.558±0.047	0.558±0.047	0.611±0.037	**0.623±0.039**	0.559±0.028	0.612±0.054
GLASS	0.704±0.087	0.704±0.087	0.704±0.087	0.732±0.064	**0.762±0.086**	0.689±0.077	0.705±0.075
GERMAN	0.379±0.050	0.379±0.050	0.379±0.050	0.411±0.036	0.437±0.040	0.410±0.038	**0.477±0.019**
YEAST1	0.525±0.026	0.525±0.026	0.525±0.026	0.533±0.036	0.525±0.021	0.513±0.037	**0.548±0.038**
HABERMAN	0.234±0.067	0.234±0.067	0.234±0.067	0.355±0.100	0.332±0.076	0.246±0.069	**0.492±0.044**
VEHICLE3	0.510±0.034	0.510±0.034	0.510±0.034	0.512±0.037	**0.561±0.033**	0.547±0.036	0.551±0.034
HAYES	0.659±0.100	0.659±0.100	0.659±0.100	**0.875±0.049**	0.766±0.083	0.768±0.140	0.787±0.100
SEGMENTATION	0.887±0.029	0.887±0.029	0.887±0.029	0.889±0.028	0.872±0.016	**0.912±0.023**	0.887±0.029
ABALONE8	0.214±0.013	0.214±0.013	0.214±0.013	**0.313±0.009**	0.238±0.011	0.219±0.017	0.311±0.020
YEAST3	0.651±0.025	0.651±0.025	0.651±0.025	0.630±0.026	**0.694±0.012**	0.636±0.010	0.659±0.023
ECOLI3	0.529±0.097	0.529±0.097	0.529±0.097	0.541±0.078	**0.612±0.058**	0.572±0.110	0.539±0.070
PAGEBLOCKS	**0.816±0.026**	**0.816±0.026**	0.816±0.026	0.811±0.024	0.813±0.044	0.815±0.032	0.812±0.022
SATIMAGE	0.676±0.036	0.676±0.036	0.676±0.036	0.680±0.034	0.688±0.027	**0.690±0.045**	0.676±0.036
YEAST-0-5-6-7-9VS4	0.409±0.110	0.409±0.110	0.409±0.110	0.497±0.041	0.519±0.073	0.455±0.150	**0.534±0.083**
LIBRAS	0.789±0.087	0.789±0.087	0.789±0.087	0.789±0.087	0.737±0.060	0.788±0.054	**0.891±0.081**
YEAST-1VS7	0.484±0.060	0.484±0.060	0.484±0.060	0.238±0.053	**0.491±0.088**	0.404±0.166	0.472±0.050
ARRYTHMIA	0.161±0.200	0.161±0.200	0.161±0.200	0.156±0.200	0.156±0.200	**0.202±0.210**	0.089±0.120
SOLAR-FLARE-M0	0.156±0.056	0.156±0.056	0.156±0.056	0.184±0.019	**0.214±0.063**	0.153±0.100	0.186±0.021
OIL	0.535±0.112	0.535±0.110	0.535±0.110	0.572±0.097	0.555±0.052	**0.613±0.120**	0.583±0.120
YEAST4	0.306±0.113	0.306±0.110	0.306±0.110	0.292±0.019	**0.414±0.040**	0.324±0.120	0.390±0.084
REDWINEQUALITY4	0.104±0.057	0.104±0.057	0.104±0.057	0.124±0.026	0.129±0.070	0.120±0.069	**0.132±0.056**
YEAST5	0.700±0.110	0.700±0.110	0.700±0.110	0.564±0.082	0.626±0.083	**0.701±0.120**	0.670±0.098
YEAST6	0.494±0.130	0.494±0.130	0.494±0.130	0.262±0.023	**0.499±0.086**	0.471±0.190	0.461±0.100
ABALONE17	0.148±0.097	0.148±0.097	0.148±0.097	0.105±0.040	0.163±0.069	0.143±0.067	**0.173±0.094**
ABALONE20	0.000±0.000	0.000±0.000	0.000±0.000	0.052±0.035	**0.066±0.067**	0.000±0.000	0.036±0.047
MEAN (K=1)	0.501	0.501	0.501	0.506	0.537	0.514	**0.544**
MEAN (K=3)	0.493	0.540	0.500	0.522	0.532	0.519	**0.558**

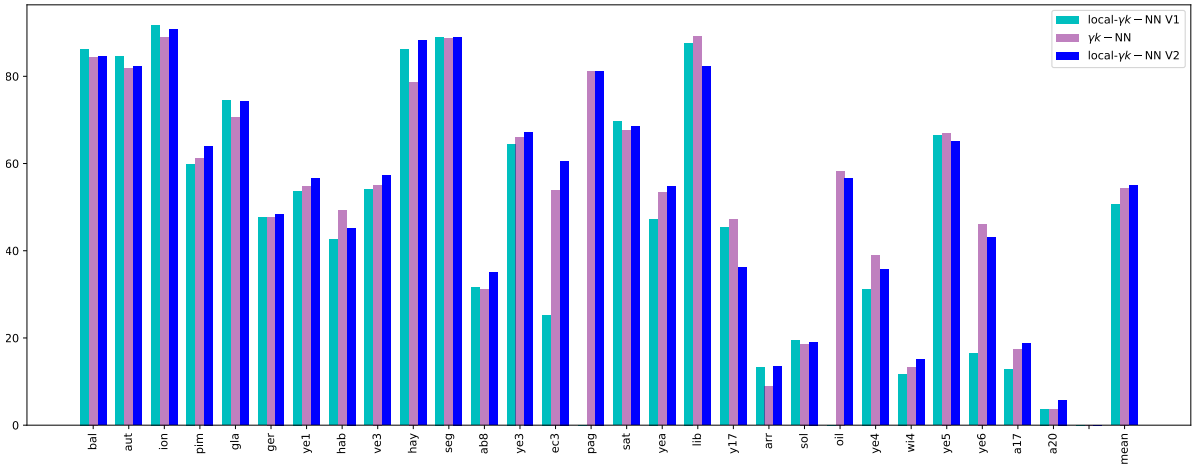

Figure 2.11: Comparison of $\gamma 1-$NN with the two versions of local-$\gamma 1-$NN, in terms of *F-measure*.

Table 2.7: Results for $k = 1$ with BA as performance measure over 5 runs. The standard deviation is indicated after the $\pm$ sign and the best results on each dataset is indicated in bold. Only the mean value when $k = 3$ is shown in the last line.

DATASETS	$k-$NN	DUP$k-$NN	W$k-$NN	CW$k-$NN	KRNN	LMNN	$\gamma k-$NN
BALANCE	0.854±0.021	0.854±0.021	0.854±0.021	0.842±0.019	**0.889±0.012**	0.855±0.041	0.842±0.019
AUTOMPG	0.816±0.061	0.816±0.061	0.816±0.061	0.811±0.052	**0.864±0.026**	0.819±0.026	0.860±0.029
IONOSPHERE	0.836±0.030	0.836±0.030	0.836±0.030	0.859±0.021	0.856±0.021	0.846±0.024	**0.912±0.036**
PIMA	0.666±0.034	0.666±0.034	0.666±0.034	0.694±0.033	**0.704±0.034**	0.667±0.017	0.697±0.037
GLASS	0.782±0.065	0.782±0.065	0.782±0.065	0.803±0.048	**0.833±0.067**	0.771±0.064	0.765±0.043
GERMAN	0.571±0.030	0.571±0.030	0.571±0.030	0.580±0.024	**0.590±0.031**	0.586±0.026	0.577±0.030
YEAST1	0.666±0.019	0.666±0.019	0.666±0.019	0.665±0.032	0.662±0.017	0.658±0.026	**0.672±0.038**
HABERMAN	0.498±0.054	0.498±0.054	0.498±0.054	0.527±0.090	0.533±0.062	0.504±0.054	**0.611±0.047**
VEHICLE3	0.673±0.023	0.673±0.023	0.673±0.023	0.674±0.028	0.712±0.027	0.701±0.032	**0.726±0.011**
HAYES	0.757±0.060	0.757±0.060	0.757±0.060	0.907±0.046	0.824±0.054	0.829±0.087	**0.919±0.042**
SEGMENTATION	0.935±0.022	0.935±0.022	0.935±0.022	0.951±0.021	0.955±0.009	0.949±0.015	**0.962±0.009**
ABALONE8	0.545±0.007	0.545±0.007	0.545±0.007	0.613±0.008	0.557±0.007	0.549±0.009	**0.626±0.018**
YEAST3	0.794±0.029	0.794±0.029	0.794±0.029	0.855±0.024	0.834±0.028	0.804±0.025	**0.857±0.029**
ECOLI3	0.744±0.069	0.744±0.069	0.744±0.069	0.823±0.059	0.812±0.052	0.733±0.035	**0.855±0.081**
PAGEBLOCKS	0.885±0.015	0.885±0.015	0.885±0.015	0.914±0.021	0.904±0.023	0.887±0.019	**0.929±0.013**
SATIMAGE	0.830±0.020	0.830±0.020	0.830±0.020	0.875±0.017	0.866±0.016	0.838±0.018	**0.891±0.012**
YEAST-0-5-6-7-9VS4	0.654±0.052	0.654±0.052	0.654±0.052	0.783±0.037	0.716±0.033	0.681±0.074	**0.793±0.036**
LIBRAS	0.839±0.050	0.839±0.050	0.839±0.050	0.839±0.050	0.834±0.048	0.839±0.047	**0.967±0.034**
YEAST-1VS7	0.717±0.032	0.717±0.032	0.717±0.032	0.675±0.077	**0.731±0.053**	0.681±0.010	0.725±0.070
ARRYTHMIA	0.575±0.160	0.575±0.160	0.575±0.160	0.568±0.170	0.572±0.160	**0.597±0.160**	0.547±0.065
SOLAR-FLARE-M0	0.551±0.025	0.551±0.025	0.551±0.025	0.651±0.018	0.582±0.029	0.550±0.043	**0.673±0.042**
OIL	0.766±0.088	0.766±0.088	0.766±0.088	0.807±0.062	0.834±0.040	0.793±0.097	**0.846±0.044**
YEAST4	0.649±0.083	0.649±0.083	0.649±0.083	0.787±0.021	0.775±0.035	0.666±0.085	**0.793±0.035**
REDWINEQUALITY4	0.535±0.027	0.535±0.027	0.535±0.027	0.587±0.039	0.557±0.044	0.543±0.032	**0.692±0.058**
YEAST5	0.872±0.074	0.872±0.074	0.872±0.074	0.914±0.055	0.909±0.057	0.862±0.067	**0.951±0.028**
YEAST6	0.777±0.100	0.777±0.100	0.777±0.100	0.849±0.093	**0.857±0.093**	0.790±0.140	0.792±0.070
ABALONE17	0.568±0.049	0.568±0.049	0.568±0.049	0.642±0.073	0.632±0.068	0.577±0.037	**0.670±0.042**
ABALONE20	0.497±0.001	0.497±0.001	0.497±0.001	0.588±0.069	0.555±0.064	0.497±0.001	**0.688±0.110**
MEAN (K=1)	0.709	0.709	0.709	0.753	0.748	0.717	**0.780**
MEAN (K=3)	0.696	0.754	0.699	0.755	0.742	0.717	**0.797**

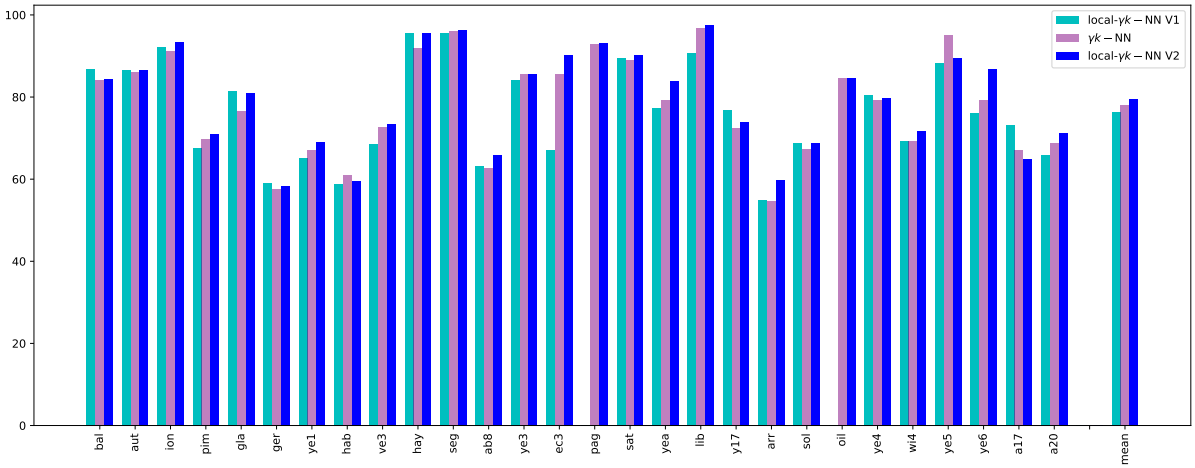

Figure 2.12: Comparison of $\gamma 1-$NN with the two versions of local-$\gamma 1-$NN, in terms of *Balanced Accuracy*.

Table 2.8: Results for $k = 1$ with GM as performance measure over 5 runs. The standard deviation is indicated after the $\pm$ sign and the best results on each dataset is indicated in bold. Only the mean value when $k = 3$ is shown in the last line.

DATASETS	$k-$NN	DUP$k-$NN	W$k-$NN	CW$k-$NN	KRNN	LMNN	$\gamma k-$NN
BALANCE	0.853±0.021	0.853±0.021	0.853±0.021	0.828±0.022	**0.888±0.012**	0.854±0.042	0.828±0.022
AUTOMPG	0.812±0.064	0.812±0.064	0.812±0.064	0.807±0.054	**0.862±0.025**	0.816±0.026	0.857±0.031
IONOSPHERE	0.821±0.035	0.821±0.035	0.821±0.035	0.849±0.023	0.847±0.023	0.841±0.022	**0.911±0.037**
PIMA	0.650±0.038	0.650±0.038	0.650±0.038	0.692±0.032	**0.703±0.034**	0.651±0.023	0.689±0.039
GLASS	0.773±0.072	0.773±0.072	0.773±0.072	0.797±0.053	**0.826±0.072**	0.761±0.068	0.777±0.060
GERMAN	0.524±0.042	0.524±0.042	0.524±0.042	0.554±0.030	**0.576±0.034**	0.551±0.032	0.570±0.030
YEAST1	0.648±0.018	0.648±0.018	0.648±0.018	0.663±0.031	0.656±0.017	0.638±0.027	**0.668±0.042**
HABERMAN	0.397±0.064	0.397±0.064	0.397±0.064	0.514±0.098	0.496±0.068	0.409±0.059	**0.556±0.079**
VEHICLE3	0.653±0.031	0.653±0.031	0.653±0.031	0.666±0.031	0.705±0.032	0.684±0.041	**0.724±0.016**
HAYES	0.718±0.086	0.718±0.086	0.718±0.086	0.902±0.051	0.806±0.066	0.803±0.119	**0.915±0.046**
SEGMENTATION	0.933±0.023	0.933±0.023	0.933±0.023	0.950±0.022	0.955±0.009	0.949±0.015	**0.962±0.009**
ABALONE8	0.431±0.016	0.431±0.016	0.431±0.016	0.596±0.009	0.472±0.016	0.436±0.021	**0.633±0.022**
YEAST3	0.774±0.039	0.774±0.039	0.774±0.039	0.852±0.029	0.823±0.036	0.789±0.034	**0.853±0.035**
ECOLI3	0.710±0.090	0.710±0.090	0.710±0.090	0.819±0.065	0.799±0.061	0.732±0.097	**0.851±0.085**
PAGEBLOCKS	0.879±0.016	0.879±0.016	0.879±0.016	0.912±0.022	0.901±0.025	0.881±0.020	**0.929±0.013**
SATIMAGE	0.819±0.023	0.819±0.023	0.819±0.023	0.873±0.018	0.862±0.017	0.832±0.027	**0.890±0.012**
YEAST-0-5-6-7-9VS4	0.569±0.092	0.569±0.092	0.569±0.092	0.775±0.044	0.671±0.047	0.589±0.142	**0.785±0.042**
LIBRAS	0.821±0.060	0.821±0.060	0.821±0.060	0.821±0.060	0.818±0.057	0.820±0.058	**0.966±0.036**
YEAST-1VS7	0.670±0.050	0.670±0.050	0.670±0.050	0.658±0.094	**0.690±0.075**	0.598±0.165	0.684±0.105
ARRYTHMIA	0.293±0.367	0.293±0.367	0.293±0.367	0.292±0.366	0.292±0.366	0.379±0.341	**0.543±0.084**
SOLAR-FLARE-M0	0.365±0.061	0.365±0.061	0.365±0.061	0.639±0.019	0.435±0.059	0.363±0.086	**0.682±0.043**
OIL	0.724±0.124	0.724±0.124	0.724±0.124	0.786±0.079	**0.823±0.050**	0.758±0.127	0.803±0.022
YEAST4	0.547±0.135	0.547±0.135	0.547±0.135	0.781±0.024	0.753±0.044	0.576±0.141	**0.782±0.043**
REDWINEQUALITY4	0.263±0.140	0.263±0.140	0.263±0.140	0.501±0.071	0.351±0.179	0.288±0.154	**0.695±0.066**
YEAST5	0.860±0.084	0.860±0.084	0.860±0.084	0.911±0.060	0.904±0.062	0.850±0.073	**0.951±0.028**
YEAST6	0.737±0.137	0.737±0.137	0.737±0.137	0.837±0.111	0.841±0.112	0.737±0.211	**0.848±0.062**
ABALONE17	0.368±0.113	0.368±0.113	0.368±0.113	0.569±0.112	0.524±0.121	0.394±0.094	**0.682±0.064**
ABALONE20	0.000±0.000	0.000±0.000	0.000±0.000	0.454±0.118	0.276±0.234	0.000±0.000	**0.663±0.126**
MEAN (K=1)	0.629	0.629	0.629	0.725	0.698	0.642	**0.775**
MEAN (K=3)	0.584	0.716	0.591	0.707	0.676	0.617	**0.789**

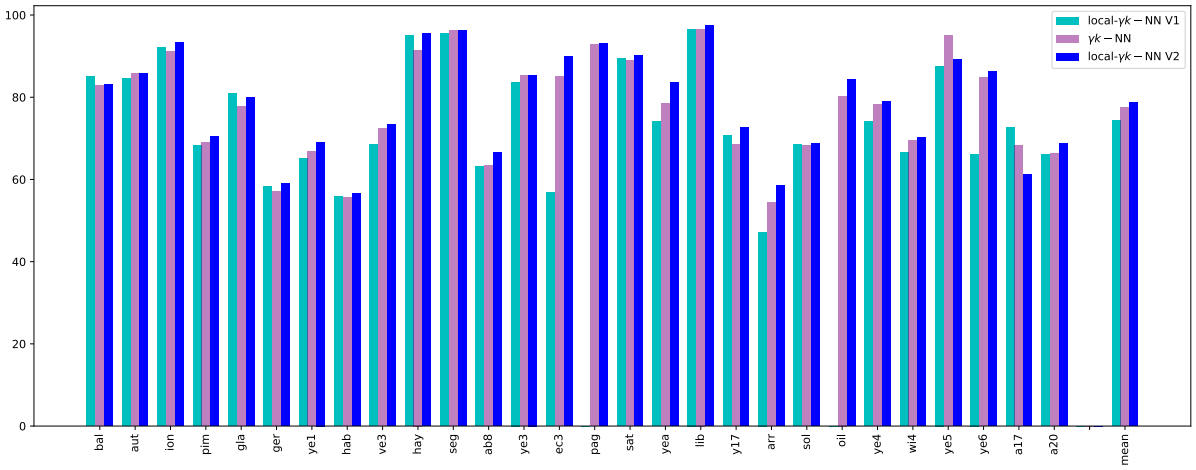

Figure 2.13: Comparison of $\gamma 1-$NN with the two versions of local-$\gamma 1-$NN, in terms of *Geometric Mean*.

Table 2.9: Results for $k = 1$ with G_1 as performance measure over 5 runs. The standard deviation is indicated after the $\pm$ sign and the best results on each dataset is indicated in bold. Only the mean value when $k = 3$ is shown in the last line.

DATASETS	k−NN	DUPk−NN	Wk−NN	CWk−NN	KRNN	LMNN	γk−NN
BALANCE	0.845±0.022	0.845±0.022	0.845±0.022	0.854±0.015	**0.884±0.012**	0.842±0.046	0.854±0.015
AUTOMPG	0.769±0.074	0.769±0.074	0.769±0.074	0.764±0.063	**0.829±0.029**	0.777±0.031	0.823±0.032
IONOSPHERE	0.814±0.039	0.814±0.039	0.814±0.039	0.842±0.030	0.838±0.029	0.820±0.031	**0.892±0.043**
PIMA	0.561±0.046	0.561±0.046	0.561±0.046	0.613±0.036	**0.626±0.038**	0.561±0.026	0.618±0.049
GLASS	0.710±0.082	0.710±0.082	0.710±0.082	0.738±0.059	**0.774±0.079**	0.696±0.076	0.685±0.050
GERMAN	0.381±0.050	0.381±0.050	0.381±0.050	0.411±0.036	0.438±0.040	0.411±0.038	**0.551±0.011**
YEAST1	0.526±0.026	0.526±0.026	0.526±0.026	0.538±0.035	0.529±0.020	0.514±0.036	**0.567±0.024**
HABERMAN	0.238±0.070	0.238±0.070	0.238±0.070	0.360±0.103	0.333±0.076	0.251±0.074	**0.514±0.049**
VEHICLE3	0.511±0.034	0.511±0.034	0.511±0.034	0.515±0.037	0.565±0.036	0.548±0.037	**0.591±0.012**
HAYES	0.689±0.086	0.689±0.086	0.689±0.086	**0.880±0.046**	0.779±0.076	0.791±0.113	0.756±0.102
SEGMENTATION	0.888±0.029	0.888±0.029	0.888±0.029	0.890±0.028	0.875±0.015	**0.909±0.029**	0.888±0.029
ABALONE8	0.214±0.013	0.214±0.013	0.214±0.013	0.331±0.009	0.239±0.012	0.219±0.017	**0.385±0.012**
YEAST3	0.654±0.023	0.654±0.023	0.654±0.023	0.649±0.018	**0.697±0.010**	0.635±0.012	0.670±0.026
ECOLI3	0.530±0.098	0.530±0.098	0.530±0.098	0.567±0.080	**0.617±0.062**	0.542±0.100	0.549±0.107
PAGEBLOCKS	**0.817±0.026**	**0.817±0.026**	**0.817±0.026**	0.813±0.025	0.813±0.044	0.815±0.032	0.812±0.022
SATIMAGE	0.677±0.036	0.677±0.036	0.677±0.036	0.690±0.032	**0.693±0.027**	0.690±0.045	0.672±0.040
YEAST-0-5-6-7-9VS4	0.422±0.104	0.422±0.104	0.422±0.104	0.519±0.043	**0.527±0.075**	0.465±0.148	0.476±0.144
LIBRAS	0.802±0.085	0.802±0.085	0.802±0.085	0.802±0.085	0.743±0.060	0.759±0.045	**0.880±0.083**
YEAST-1VS7	0.487±0.058	0.487±0.058	0.487±0.058	0.292±0.073	0.493±0.088	**0.500±0.125**	0.479±0.071
ARRYTHMIA	0.171±0.217	0.171±0.217	0.171±0.217	0.167±0.214	0.167±0.214	**0.205±0.220**	0.171±0.217
SOLAR-FLARE-M0	0.165±0.068	0.165±0.068	0.165±0.068	0.243±0.017	0.215±0.064	0.117±0.079	**0.267±0.032**
OIL	0.546±0.108	0.546±0.108	0.546±0.108	0.580±0.094	0.571±0.048	**0.666±0.100**	0.597±0.101
YEAST4	0.312±0.116	0.312±0.116	0.312±0.116	0.357±0.021	**0.436±0.041**	0.350±0.132	0.402±0.087
REDWINEQUALITY4	0.108±0.058	0.108±0.058	0.108±0.058	0.152±0.037	0.133±0.071	0.122±0.070	**0.192±0.055**
YEAST5	0.705±0.110	0.705±0.110	0.705±0.110	0.602±0.081	0.649±0.081	**0.720±0.074**	0.697±0.096
YEAST6	0.500±0.139	0.500±0.139	0.500±0.139	0.353±0.053	**0.529±0.099**	0.466±0.181	0.479±0.115
ABALONE17	0.150±0.097	0.150±0.097	0.150±0.097	0.150±0.057	**0.183±0.078**	0.170±0.072	0.147±0.024
ABALONE20	0.000±0.000	0.000±0.000	0.000±0.000	**0.083±0.052**	0.076±0.075	0.000±0.000	0.012±0.024
MEAN (K=1)	0.507	0.507	0.507	0.527	0.545	0.520	**0.558**
MEAN (K=3)	0.508	0.552	0.514	0.540	0.546	0.527	**0.559**

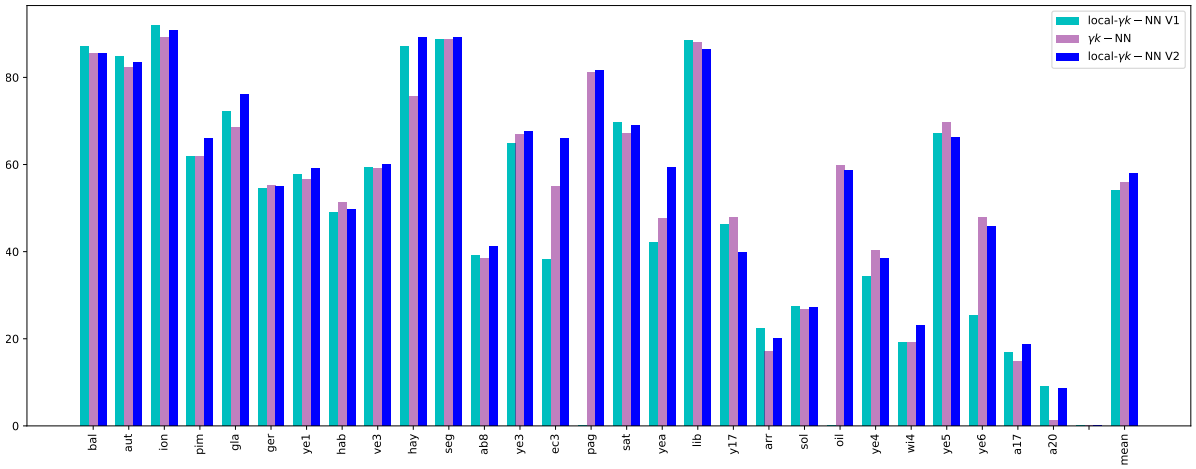

Figure 2.14: Comparison of $\gamma 1-$NN with the two versions of local-$\gamma 1-$NN, in terms of *G-measure*.

Chapter 3

Metric Learning from Few Positives

This chapter is based on the following publications

Rémi Viola, Rémi Emonet, Amaury Habrard, Guillaume Metzler and Marc Sebban. Learning from Few Positive: a Provably Accurate Metric Learning Algorithm to Deal with Imbalanced Data. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2020, Virtual, Japan [118].

Rémi Viola, Rémi Emonet, Amaury Habrard, Guillaume Metzler and Marc Sebban. MLFP: Un Algorithme d'apprentissage de Métrique pour la Classification de données déséquilibrées. In *Conférence sur l'Apprentissage automatique (CAp)*, 2020, Virtual, France [115].

Abstract

As explained before, learning from imbalanced data, where the positive examples are very scarce, remains a challenging task from both a theoretical and algorithmic perspective. In this chapter, we extend the idea presented in our first contribution by using a *metric learning* strategy. Unlike the state-of-the-art metric learning methods, our algorithm **MLFP**, for *Metric Learning from Few Positives*, learns a new representation that is used only when a test query is compared to a minority training example. From a geometric perspective, in this projection space, the idea is still to artificially bring positive examples closer to the query without changing the distances to the negative (majority class) data. This strategy allows us to expand the decision boundaries around the positives, yielding a better $F\text{-}Measure$. Beyond the algorithmic contribution provided by **MLFP**, this chapter presents generalization guarantees on the False Positive and False Negative rates. Extensive experiments conducted on several public and private imbalanced datasets show the effectiveness of our method.

3.1 Introduction

Fraud detection in bank or insurance applications [2, 99], and anomaly identification for medical diagnosis [3] are some societal challenges requiring to address the problem of learning from highly imbalanced data. When dealing with such a setting, one has to face two major issues: (i) the scarcity of the class of interest, only composed of a few positive data, which limits the efficiency of standard margin-based loss functions; (ii) the scattering of positive examples in the total mass of the training data, which makes the estimation of local densities much more complicated than in balanced scenarios.

As we said in Chapter 1, several solutions have been proposed in the literature to address these two problems. Most of them consist in applying sampling strategies which aim to balance the dataset by reducing the number of negative examples and/or creating new synthetic positive

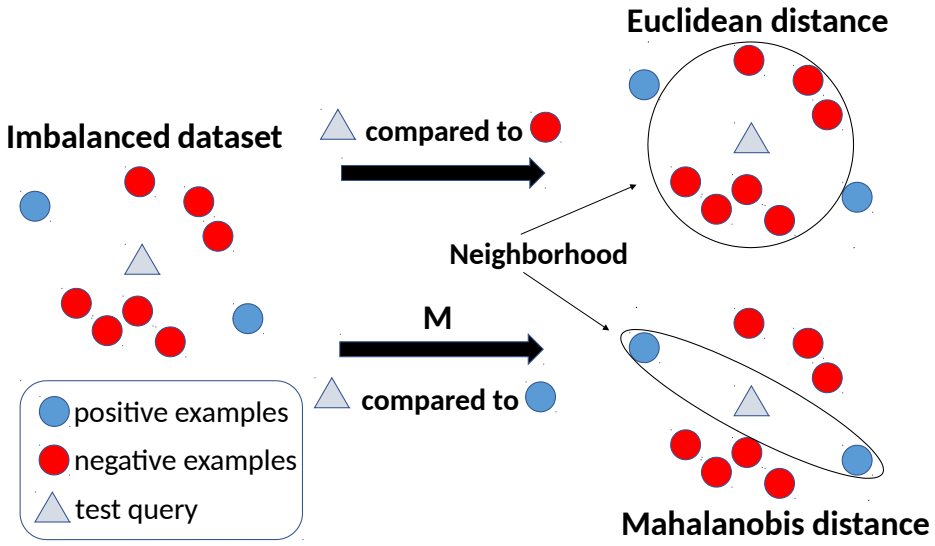

Figure 3.1: Intuition behind our method **MLFP**: a PSD matrix $\mathbf{M}$ is optimized under constraints, and is used only when a test query is compared to a positive example. The distance to the negative examples is kept unchanged. This allows the learned metric to expand the decision boundaries around the positives and thus to capture more examples of the class of interest.

data [102, 88]. On the other hand, one can resort to cost-sensitive algorithms [63] which assign a weight to each class (or even to each example) so that the classifier can focus better on the minority class. Other strategies include the use of ensemble methods [126, 44] or the specific adaptation of existing approaches such as deep learning [61, 35] or kernel methods [81, 32, 132].

In this chapter, we address the problem of learning from imbalanced data from a metric learning perspective [9, 67]. Learning a metric specifically designed for the application at hand may present several advantages in the context of imbalanced datasets: (i) the metric can be learned under semantic constraints allowing us to expand the decision boundaries around the positives; (ii) this framework enables to design optimization problems based on the geometry of the data without suffering from the issues of standard accuracy-based loss functions (*e.g.*, hinge loss for SVMs, exponential loss for boosting, logistic loss for logistic regression); (iii) metric learning is a nice setting to derive theoretical guarantees on the learned transformation [10]. Surprisingly, despite these interesting features, metric learning has not received much attention to address the problem of learning from imbalanced data (see, *e.g.*, the recent papers [39], [119] and [51]). The goal of this chapter is to bridge this gap from both an algorithmic and a theoretical perspective. As illustrated in Figure 3.1, we propose the algorithm **MLFP** that optimizes a linear transformation (via a *Positive Semi Definite* (PSD) matrix $\mathbf{M}$ of a Mahalanobis distance) only when a test query is compared to a minority training example. A single metric $\mathbf{M}$ is learned for the whole space taking the geometry of the data into account. Unlike the standard metric learning algorithms (see, *e.g.*, **LMNN** [123] or **ITML** [30]), our method boils down to artificially bringing positive examples closer to the query without challenging the features of the negatives. This has a direct impact on the decision boundaries around the positives allowing us to capture more examples of the class of interest yielding a better $F\text{-}Measure$ (see Section 3.3 for a formal definition). By using the Uniform Stability Framework, we derive theoretical guarantees on the learned matrix $\mathbf{M}$ showing the actual capability of **MLFP** to control the false positive and false negative rates.

The chapter is organized as follows. In Section 3.2, we report some related work on metric learning for imbalanced data classification. Section 3.3 is dedicated to the presentation of our metric learning algorithm **MLFP**. Section 3.4 presents a theoretical analysis using the Uniform

Stability Framework and Section 3.5 illustrates the performance of **MLFP** compared to state-of-the-art algorithms.

3.2 Related Work

Most of the metric learning algorithms (see [9, 67] for a survey) are based on the optimization of the Mahalanobis distance between two points $\mathbf{x}_i$ and $\mathbf{x}_j \in \mathbb{R}^d$:

$$\mathcal{D}_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j)^2 = (\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{M} (\mathbf{x}_i - \mathbf{x}_j),$$

where $\mathbf{M}$ is a $d \times d$ *Positive Semi Definite* matrix. One can express $\mathbf{M}$ as $\mathbf{L}^T\mathbf{L}$ where $\mathbf{L}$ is a $r \times d$ matrix where r is the rank of $\mathbf{M}$. Thus, this distance can be seen as the Euclidean distance in a new feature space $\mathbf{Lx}$.

As explained in Section 1.4, a well-known representative of this family of algorithms is the *Large Margin Nearest Neighbor* (**LMNN**) [123]. For each example of a training set of size m, the learned metric $\mathbf{M}$ aims to bring closer the neighbors of the same class (called target neighbors) while pushing away the examples of other classes (the impostors). While the number of constraints is in the order of km^2 with k the number of considered nearest neighbors, the authors proposed an efficient subgradient descent algorithm which benefits from the fact that many of these constraints are trivially satisfied. This algorithm has been shown to be very efficient and to scale well with large datasets. However, it is worth noticing that **LMNN** is not designed to take into account some imbalance in the data. Indeed, the similarity constraints constructed from pairs of examples of the same class do not make any difference between the positive and negative examples. Therefore, in imbalanced scenarios, **LMNN** is prone to focus on the majority class and thus is subject to miss the positive examples.
This remark also holds for *Information Theoretic Metric Learning* (**ITML**) [30] or *Geometric Mean Metric Learning* (**GMML**) [129]. Like **LMNN**, they work with pairs of examples. The main difference for **ITML** comes from the use of a *LogDet* regularization which constrains $\mathbf{M}$ to remain close to some prior matrix $\mathbf{M}_0$. It can be shown that **ITML** provides a cheap way to fulfill the PSD constraint on $\mathbf{M}$. On the other hand, **GMML** learns a matrix $\mathbf{M}$ to compute the distance between the similar examples and the matrix $\mathbf{M}^{-1}$ to compute the dissimilar ones. The idea is that if $\mathbf{M}$ brings similar examples close to each other (*i.e.* all the eigenvalues are less than one), then $\mathbf{M}^{-1}$ will push dissimilar examples away. Built on a geometric intuition, they learn the metric with a convex optimization problem which has a closed form solution by seing the latter as an optimization problem on the Riemannian manifold of PSD matrices.

The first attempts to address the problem of learning a metric from imbalanced datasets have been proposed very recently. [119] introduce an *iterative metric learning* algorithm (**IML**) that aims to define a stable neighborhood used to predict the label of a new test data. The method repeats two main steps: (i) the learning of a linear transformation, *e.g.*, by using **LMNN**, and (ii) a training sample selection given a test example. The procedure is repeated until stabilization of the neighborhood. By repeating the process several times, **IML** is able to locally separate positives from negatives. However, the main issue comes from the algorithmic complexity of the method, which requires to apply **LMNN** and to update the pairs used for the training process at each iteration.

Another approach to learn metrics from imbalanced datasets has been recently proposed [51]. In their *Imbalanced Metric Learning* algorithm (**ImbML**), the authors take into account the nature of the pairwise constraints by using two different sub-losses, one for each label, weighted according to the number of positive and negative examples respectively. This intuitive and natural way to proceed prevents the algorithm from favoring the majority class. However, we will see that applying the learned metric $\mathbf{M}$ to all examples is not necessary, focusing only on

the minority class appears to be much more efficient and allows us notably to better control the false negatives.

Finally, [39] introduce **DMBK** for *Distance Metric by Balancing KL-divergence*. This algorithm resorts to the KL-divergence to represent normalized between-class divergences. Combined with a geometric mean, **DMBK** is able to make these divergences balanced. Note that this method makes sense in the multi-class setting, but is meaningless for addressing binary problems, due to the use of the normalization while computing the KL-divergence.

Beyond the algorithmic limitations of the previous state-of-the-art algorithms, note that none of them comes with guarantees on the classification error. In this chapter, we address this problem by studying the capability of **MLFP** to optimize a metric $\mathbf{M}$ which provides a good compromise between (i) expanding the decision boundaries around the positives which enables to reduce the false negative rate at test time (one of the main issues faced in imbalanced learning); (ii) controlling this expansion to prevent the algorithm from detecting too many false alarms, represented by the false positive rate. The theoretical results take the form of guarantees on the learned metric using the uniform stability framework [13] which measures the stability of the output of the algorithm when the training set is subject to slight changes.

3.3 Metric Learning for Imbalanced Data

In this section, we present our algorithm **MLFP**, for *Metric Learning from Few Positives*. Let us remind that we denote by $S = \{\mathbf{z}_i = (\mathbf{x}_i, y_i)\}_{i=1}^m$ the set of m training examples drawn *i.i.d.* from an unknown joint distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, where $\mathbf{x}_i \in \mathcal{X}$ (here $\mathcal{X} = \mathbb{R}^d$) is a feature vector and $y_i \in \mathcal{Y}$ (here $\mathcal{Y} = \{-1, +1\}$) corresponds to its associated label. The label $+1$ is used to denote the positive or the minority class. We further note $S = S_+ \cup S_-$ with S_+ the set of m_+ positive examples and S_- the set of m_- negative examples, such that $m = m_+ + m_-$.

3.3.1 Problem Formulation

In our approach, we use the Euclidean distance when comparing a query point to a majority-class example. The originality comes from the use of an optimized Mahalanobis distance when comparing a query to a minority-class sample. The objective of this strategy is to formulate a metric learning problem leading to a classifier (a $k-$NN here) which is accurate on both classes even in an imbalanced scenario.

In order to avoid the pitfall of classic metric learning algorithms that are prone to focus on the majority class, we propose to give more importance to the minority class composed of the positive instances. Our algorithm **MLFP** tries to control the false positive (FP) and false negative (FN) rates thanks to the following constrained optimization problem:

$$\min_{\mathbf{M}\in\mathbb{S}^+} \quad \frac{1}{m^3}\left((1-\alpha)\sum_{\substack{(\mathbf{x}_i,\mathbf{x}_j,\mathbf{x}_k)\\ y_i=y_j\neq y_k=-1}} \ell_{\mathrm{FN}}(\mathbf{M},\mathbf{z}_i,\mathbf{z}_j,\mathbf{z}_k) + \alpha \sum_{\substack{(\mathbf{x}_i,\mathbf{x}_j,\mathbf{x}_k)\\ y_i=y_j\neq y_k=1}} \ell_{\mathrm{FP}}(\mathbf{M},\mathbf{z}_i,\mathbf{z}_j,\mathbf{z}_k)\right) + \mu\|\mathbf{M}-\mathbf{I}\|_{\mathcal{F}}^2$$

$$\text{such that } \lambda_{\max}(\mathbf{M}) \leq 1. \tag{3.1}$$

where $\mathbb{S}^+$ is the set of PSD matrices, $\lambda_{\max}(\mathbf{M})$ is the largest eigenvalue of the PSD matrix $\mathbf{M}$, ℓ_{FN} and ℓ_{FP} are defined by:

$$\ell_{\mathrm{FN}}(\mathbf{M},\mathbf{z}_i,\mathbf{z}_j,\mathbf{z}_k) = [1 - c + \mathcal{D}_{\mathbf{M}}(\mathbf{x}_i,\mathbf{x}_j)^2 - \mathcal{D}(\mathbf{x}_i,\mathbf{x}_k)^2]_+,$$

$$\ell_{\text{FP}}(\mathbf{M}, \mathbf{z}_i, \mathbf{z}_j, \mathbf{z}_k) = [1 - c + \mathcal{D}(\mathbf{x}_i, \mathbf{x}_j)^2 - \mathcal{D}_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_k)^2]_+,$$

where $[a]_+ = \max(0, a)$, α is the positive rate $\frac{m_+}{m}$ and $\mu\|\mathbf{M} - \mathbf{I}\|_{\mathcal{F}}^2$ is a regularization term which penalizes a large deviation from the Euclidean distance. The hyper-parameter c controls the margin we want to preserve between pairs of dissimilar examples according to the Euclidean space and the learned one.

Problem (3.1) is composed of two terms where triplets are involved. Unlike standard metric learning algorithms, our method takes into account both the Euclidean distance $\mathcal{D}$ and the metric learned $\mathcal{D}_M$. More precisely: the first term ℓ_{FN} aims to gather the minority class examples with respect to the learned metric such that the distance between two positives (using $\mathbf{M}$) is less than the distance to a negative example (using the Euclidean distance). This subloss can be seen as a way to prevent the model from generating false negatives (FN). The second term ℓ_{FP} works in a similar manner. The only difference lies in the fact that the query $\mathbf{x}_i$ is a negative example. Thus, we learn $\mathbf{M}$ such that the positive queries $\mathbf{x}_k$ are not brought too close to $\mathbf{x}_i$, *i.e.* the Euclidean distance between two negatives $\mathbf{x}_i$ and $\mathbf{x}_j$ (with respect to the Euclidean distance) is lower than the distance between $\mathbf{x}_i$ and $\mathbf{x}_k$ (with respect to $\mathbf{M}$). This subloss can be seen as a way to prevent the model from generating false positives (FP).

As presented in Section 1.5.1, minimizing the *F-Measure* boils down to finding a good trade-off between FP and FN. However, in a highly imbalanced setting, where the number of positives is very low, missing only a few positives leads to a dramatic decrease of the *F-Measure*. That is why we constrain in Problem (3.1) the largest eigenvalue $\lambda_{\max}(\mathbf{M})$ to be lower than 1, so that the learned matrix $\mathbf{M}$ aims to pay more attention to the positive class. In the next section, we provide a formal explanation of its use.

3.3.2 On the Impact of the Constraint

We study here the impact of the $\lambda_{\max}(\mathbf{M})$ value on both FN and FP and, thus the influence of the constraint in the optimization problem.

Proposition 3. *Let $\mathbb{P}[FN_{\mathbf{M}}(\mathbf{x})]$ (resp. $\mathbb{P}[FP_{\mathbf{M}}(\mathbf{x})]$) be the probability of a positive query (resp. a negative query) $\mathbf{x}$ of being a false negative (resp. a false positive) using the $1-NN$ algorithm with the learned matrix $\mathbf{M}$ and $\mathbb{P}[FN(\mathbf{x})]$ (resp. $\mathbb{P}[FP(\mathbf{x})]$) the same probability using the Euclidean distance.*
Then, if $\lambda_{\max}(\mathbf{M}) \leq 1$, we have:

$$\mathbb{P}[FN_{\mathbf{M}}(\mathbf{x})] \leq \mathbb{P}[FN(\mathbf{x})] \text{ and } \mathbb{P}[FP_{\mathbf{M}}(\mathbf{x})] \geq \mathbb{P}[FP(\mathbf{x})].$$

Sketch of proof. Let ε be the distance from $\mathbf{x}$ to its nearest neighbor $N_{\mathbf{x}}$. The example $\mathbf{x}$ is a false negative if $N_{\mathbf{x}} \in S_-$, that is, all positives $\mathbf{x}' \in S_+$ are outside an ellipsoid $\mathcal{E}_{\varepsilon,\mathbf{M}^{-1}}(\mathbf{x})$, defined by ε and $\mathbf{M}$. Therefore, we have:

$$\mathbb{P}[FN_{\mathbf{M}}(\mathbf{x})] = (1 - \mathbb{P}\left[\mathbf{x}' \in \mathcal{E}_{\varepsilon,\mathbf{M}^{-1}}(\mathbf{x})\right])^{m_+}. \tag{3.2}$$

When the Euclidean distance is used, we deal with a standard sphere $\mathcal{S}_\varepsilon$ of radius ε, and we get:

$$\mathbb{P}[FN(\mathbf{x})] = (1 - \mathbb{P}\left[\mathbf{x}' \in \mathcal{S}_\varepsilon(\mathbf{x})\right])^{m_+}. \tag{3.3}$$

Having $\lambda_{\max}(\mathbf{M}) \leq 1$ implies Eq. (3.2) $\leq$ Eq. (3.3). Indeed $\lambda_{\max}(\mathbf{M}) \leq 1$ implies that the sphere $\mathcal{S}_\varepsilon$ is included in the ellipsoid $\mathcal{E}_{\varepsilon,\mathbf{M}^{-1}}$ as illustrated in Figure 3.2. By this choice, we expand the decision boundaries around positives and thus capture more minority class examples.

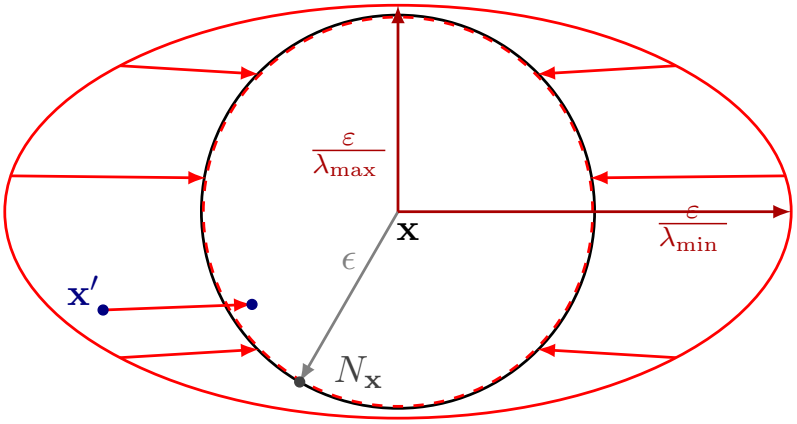

Figure 3.2: Illustration of the constraint $\lambda_{\max}(\mathbf{M}) \leq 1$. Without learning the matrix $\mathbf{M}$, the Euclidean distance is used both to compare a query $\mathbf{x}$ to a negative $N_{\mathbf{x}}$ and to a positive $\mathbf{x}'$. The isodistance curves are thus spherical and identical (one in solid black for $N_{\mathbf{x}}$, one in dashed red for $\mathbf{x}'$). By learning the matrix $\mathbf{M}$, we virtually change the distance of the query to the positive examples. The isodistance curves for the positives are now ellipses, like the one represented in red. In the example, the positive $\mathbf{x}'$, that is outside the sphere, is inside the ellipse and will thus be considered closer, with the constraint $\lambda_{\max}(\mathbf{M}) \leq 1$, than the negative $N_{\mathbf{x}}$ that lies on the black sphere. With this same constraint, we are sure that the ellipse is enclosing the circle (*i.e.* $\frac{\varepsilon}{\lambda_{\max}} \geq \varepsilon$) and so that all positives will be brought closer to the query. In the end, this constraint ensures that we increase the influence of the positives and thus leads to the decrease of FN.

Using a similar scheme, we can prove the second inequality of Proposition 3. When $\mathbf{x}$ is negative and $N_{\mathbf{x}} \in S_+$, we have:

$$\mathbb{P}[FP_{\mathbf{M}}(\mathbf{x})] = (1 - \mathbb{P}\left[\mathbf{x}' \in \mathcal{E}_{\varepsilon,\mathbf{M}^{-1}}(\mathbf{x})\right])^{m_-}, \tag{3.4}$$

and

$$\mathbb{P}[FP(\mathbf{x})] = (1 - \mathbb{P}\left[\mathbf{x}' \in \mathcal{S}_{\varepsilon}(\mathbf{x})\right])^{m_-}. \tag{3.5}$$

□

By looking at Equations (3.2) and (3.4), we can note that they are both exponentially decreasing *w.r.t.* to the number of positives and negatives respectively. However, in imbalanced scenarios, the number of negatives is supposed to be much higher than the number of positives. Thus, the probability of having a false positive is decreasing faster than the probability of having a false negative. We then choose to learn a matrix $\mathbf{M}$ under the constraint $\lambda_{\max}(\mathbf{M}) \leq 1$, so that our algorithm will focus first on reducing FN. An illustration of the impact of this constraint in terms of decision boundaries is shown in Figure 3.3. The experiments in Section 3.5 will confirm that the use of this constraint is very relevant from an *F-Measure* perspective and is able to reduce the number of FN at test time.

3.4 Theoretical Analysis

In this section, we provide generalization guarantees about the learned metric $\mathbf{M}$ using the Uniform Stability framework [13] adapted to metric learning [10]. Then, we use this result to derive classification guarantees over a 1−Nearest Neighbor (1−NN) classifier making use of this

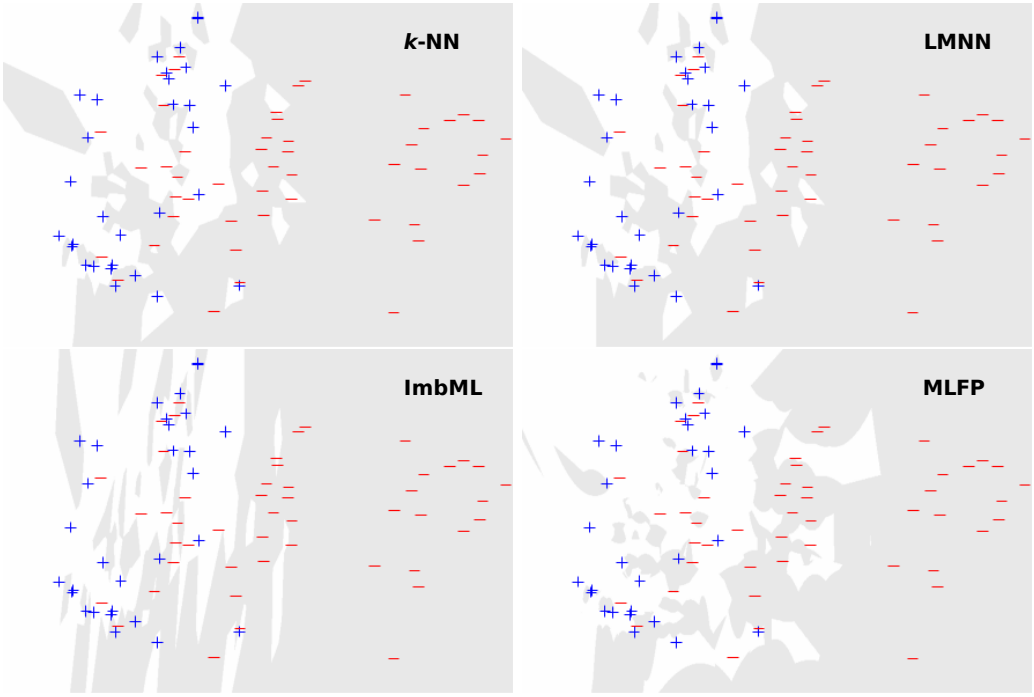

Figure 3.3: Illustration of the impact of the constraint $\lambda_{\max}(\mathbf{M}) \leq 1$ in **MLFP** (bottom right) compared to $k-$**NN** (top left), **LMNN** (top right), **ImbML** (bottom left) on the *autompg* dataset with a 1−NN classifier. We perform a PCA, keeping the two most relevant dimensions, and plot the test set on a mesh grid of the space. In light grey (*resp.* white), areas classified as negative (*resp.* positive).

metric. Note that the whole study is conducted under the constraint $\lambda_{\max}(\mathbf{M}) \leq 1$ as used in Problem (3.1).

First, we denote by ℓ the weighted combination of ℓ_{FN} and ℓ_{FP} as defined in Problem (3.1) and F_S the objective function to optimize over the training set $S = \{\mathbf{z}_i\}_{i=1}^m$. We have:

$$F_S = \frac{1}{m^3} \sum_{i,j,k=1}^{m} \ell(\mathbf{M}, (\mathbf{z}_i, \mathbf{z}_j, \mathbf{z}_k)) + \mu \|\mathbf{M} - \mathbf{I}\|_{\mathcal{F}}^2.$$

Let $\mathcal{R}_S$ be the associated empirical risk over S defined as:

$$\mathcal{R}_S = \frac{1}{m^3} \sum_{i,j,k=1}^{m} \ell(\mathbf{M}, (\mathbf{z}_i, \mathbf{z}_j, \mathbf{z}_k)),$$

and $\mathcal{R}$ be the corresponding expected true risk defined as:

$$\begin{aligned} \mathcal{R} = \mathop{\mathbb{E}}_{S \sim \mathcal{D}^m} [\mathcal{R}_S] &= \mathop{\mathbb{E}}_{S \sim \mathcal{D}^m} \left[\frac{1}{m^3} \sum_{i,j,k=1}^{m} \ell(\mathbf{M}, (\mathbf{z}_i, \mathbf{z}_j, \mathbf{z}_k)) \right] \\ &= \mathop{\mathbb{E}}_{\mathbf{z},\mathbf{z}',\mathbf{z}'' \sim \mathcal{D}} \left[\ell(\mathbf{M}, (\mathbf{z}, \mathbf{z}', \mathbf{z}'')) \right]. \end{aligned}$$

The last equality is due to the *i.i.d.* aspect of the expectation.

3.4.1 Uniform Stability

Intuitively, an algorithm is stable if its output, in terms of loss, does not change significantly under a small modification of the training sample. The supremum of this change must be

bounded in $\mathcal{O}(1/m)$.

Definition 4. *A learning algorithm $\mathcal{A}$ has a uniform stability in $\frac{\kappa}{m}$ with respect to a loss function ℓ and parameter set θ, with κ a positive constant if:*

$$\forall S,\ \forall i,\ 1 \leq i \leq m,\ \sup_{Z} |\ell(\theta_S, Z) - \ell(\theta_{S^i}, Z)| \leq \frac{\kappa}{m},$$

where S is a learning sample of size m, $Z = (\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3) = ((\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_3, y_3))$ is a triplet of labeled examples, θ_S the model parameters learned from S, θ_{S^i} the model parameters learned from the sample S^i obtained by replacing the i^{th} example $\mathbf{z}_i$ from S by another example $\mathbf{z}'_i$ independent from S and drawn from $\mathcal{D}$. Finally, $\ell(\theta_S, Z)$ is the loss suffered at Z.

In this definition, S^i represents the notion of small modification of the training sample. The next definition aims to study the evolution of the loss function according to the considered triplets Z and Z'.

Definition 5. *A loss function ℓ is said to be σ-admissible, w.r.t. the distance metric $\mathbf{M}$ if (i) it is convex w.r.t. its first argument and (ii) if the following condition holds:*

$$\forall Z, Z' \quad |\ell(\mathbf{M}, Z) - \ell(\mathbf{M}, Z')| \leq \sigma,$$

where $Z = (\mathbf{z}_i, \mathbf{z}_j, \mathbf{z}_k)$ and $Z' = (\mathbf{z}'_i, \mathbf{z}'_j, \mathbf{z}'_k)$ are two triplets from a sample S and drawn from $\mathcal{D}$.

From the two above definitions, if we also suppose that for all $\mathbf{x}$, we have $\|\mathbf{x}\| \leq K$, we can state the following generalization bound.

Theorem 2[1]**.** *Let $\delta > 0$ and $m > 2$. Let S be a sample of m randomly selected training examples. Let $\mathbf{M}$ be the matrix learned from Problem (3.1) which has a uniform stability in $\frac{\kappa}{m}$. The loss function ℓ as defined above is σ-admissible. With probability $1-\delta$, the following bound on the true risk $\mathcal{R}$ of ℓ holds:*

$$\mathcal{R} \leq \mathcal{R}_S + 2\frac{\kappa}{m} + (2\kappa + 2\sigma)\sqrt{\frac{\ln(2/\delta)}{2m}},$$

where

$$\kappa = \frac{12}{\mu} \times ((1-\alpha)K^2)^2 \text{ and } \sigma = (1-\alpha)(1-c+4K^2).$$

The derived bound provides guarantees on the generalization performances of the learned metric on the distribution $\mathcal{D}$ *w.r.t.* to the loss ℓ. We now make use of this bound to provide classification guarantees of a $1-$NN making use of the learned metric $\mathbf{M}$.

3.4.2 Classification Guarantees

We derive here generalization guarantees on the FP and FN rates for a $1-$NN classifier making use of the metric $\mathbf{M}$ learned by **MLFP**. Let S be the learning sample of size m used by a nearest-neighbor classifier. Let us define the empirical risks for FP and FN:

$$\mathcal{R}_{FP}(S) = \mathbb{E}_{\mathbf{z}=(\mathbf{x},y)\sim D} \mathbb{1}_{\{\mathcal{D}_{\mathbf{M}}(\mathbf{x},\mathbf{x}_p)^2 \leq \mathcal{D}(\mathbf{x},\mathbf{x}_n)^2\}} \times \mathbb{1}_{\{y=-1\}}.$$

where $\mathbf{x}_p, \mathbf{x}_n \in S$ are respectively the nearest positive and negative neighbors of $\mathbf{x}$ in S. Symmetrically, we have:

$$\mathcal{R}_{FN}(S) = \mathbb{E}_{\mathbf{z}=(\mathbf{x},y)\sim D} \mathbb{1}_{\{\mathcal{D}(\mathbf{x},\mathbf{x}_n)^2 \leq \mathcal{D}_{\mathbf{M}}(\mathbf{x},\mathbf{x}_p)^2\}} \times \mathbb{1}_{\{y=1\}}.$$

We consider then the expected true risks averaged over all the training samples of size m:

$$\mathcal{R}_{FP} = \mathbb{E}_{S\sim D^m} \mathcal{R}_{FP}(S) \text{ and } \mathcal{R}_{FN} = \mathbb{E}_{S\sim D^m} \mathcal{R}_{FN}(S).$$

We can now introduce our main result.

[1] The elements of the proof are given at the end of this Chapter, in Section 3.7.

Theorem 3[2]**.** *Let $\delta > 0$ and $m > 0$. Let S be a training sample of size m i.i.d. from a distribution D, z a new instance i.i.d. from D, and let $\mathbf{M}$ be the learned matrix from Problem* (3.1) *which has a uniform stability in $\frac{\kappa}{m}$ with respect to the loss ℓ. Considering that the loss function ℓ is σ-admissible, let us denote by $\mathcal{R}_S$ its empirical risk. With probability $1-\delta$, we have the following bounds for the FP and FN rates:*

$$\mathcal{R}_{FP} \leq \frac{1}{\alpha}\left[\mathcal{R}_{S\cup\{z\}} + \frac{2\kappa}{m+1} + (2\kappa + 2\sigma)\sqrt{\frac{\ln(2/\delta)}{2(m+1)}}\right],$$

$$\mathcal{R}_{FN} \leq \frac{1}{1-\alpha}\left[\mathcal{R}_{S\cup\{z\}} + \frac{2\kappa}{m+1} + (2\kappa + 2\sigma)\sqrt{\frac{\ln(2/\delta)}{2(m+1)}}\right].$$

By comparing these two bounds, one can observe that when the class imbalance becomes important, *i.e.* when α takes a low value, the guarantees on the FN rate become better than the guarantees on FP. This result provides a theoretical confirmation that our approach - thanks to the constraint $\lambda_{\max}(\mathbf{M}) \leq 1$ - is able to focus more on reducing FN. An illustration of this phenomenon will be shown in the next section.

3.5 Experiments

In this section, we compare **MLFP** to other metric learning algorithms, focusing on (highly) imbalanced datasets. For all experiments, as done in both [119] and [123], we use a 3−Nearest Neighbor classifier. The results of the experiments with a 1−Nearest Neighbor classifier are available at the end of this Chapter. Note that the source code of **MLFP**, allowing the interested reader to reproduce these experiments is available[3].

3.5.1 Experimental Setup

We use several public datasets from the UCI[4] and KEEL[5] repositories. These datasets are diverse in terms of imbalance ratio (IR, number of majority examples per positive example), dimension, number of examples, as shown in Table 2.2. In addition, we use eight datasets provided by the French Ministry of the Economy and Finances (DGFiP). We remind the reader that it corresponds to the tax returns of French companies and are used for fraud detection. These frauds may correspond to overvalued charges, voluntary reductions in profits or, local or international, VAT fraud. As the DGFiP can only control a small part of the 3,000,000 companies, it is essential to optimise the selection of these companies in order to reduce errors as much as possible. Considering the control process, it is less costly for the DGFiP to quickly control a non-fraudulent company than to let a big fraudster pass. It is therefore important for it to reduce the number of FN, even if it means increasing the number of FP a little.

All the datasets are standardized by substracting the mean and dividing by the standard deviation. We use the *F-Measure* as the performance criterion to compare the different methods. Furthermore, 80% of the dataset is randomly selected in order to train the model and 20% to test it. The different hyper-parameters are tuned with a 10-fold-cross-validation over the training set. The sampling of the test set is repeated 5 times and we report the average results in terms of *F-Measure* (F_1).

For our **MLFP** method, the hyper-parameters μ for the regularization and c for the margin are both tuned in the range $[0, 1]$, using a Bayesian optimization with 400 calls. The Bayesian

[2]The elements of the proof are given at the end of this Chapter, in Section 3.7.
[3]`https://github.com/RemiViola/MLFP`
[4]`https://archive.ics.uci.edu/ml/datasets.html`
[5]`https://sci2s.ugr.es/keel/datasets.php`

optimization is done with the Scikit-Optimize library[6]. As the matrix $\mathbf{M}$ can be expressed as $\mathbf{L}^T\mathbf{L}$ (Cholesky decomposition), we directly learn a diagonal matrix $\mathbf{L}$. Since we are not particularly interested, in this chapter, in low rank matrices, we do not impose any constraint on the dimension of $\mathbf{L}$. At each iteration of the optimization process, the spectral radius of the matrix $\mathbf{L}$ is constrained to be less than one so that $\mathbf{M} = \mathbf{L}^T\mathbf{L}$ has its largest value less than one.

We compare **MLFP** with several methods:

- The 3−Nearest Neighbor algorithm **(3−NN)**, as a baseline.
- **LMNN**, where the hyper-parameter μ, which controls the trade-off between the two parts of the loss (see [123] for more details), is tuned in $[0, 1]$ using a Bayesian optimization with 20 calls.
- **ITML** [30].
- **GMML** [129], where the parameter t is tuned in $[0, 1]$ also with 20 calls of a Bayesian optimization.
- **IML** [119] where we select $5*k = 15$ points for the sampling selection and we also tune the hyper-parameter of the **LMNN** algorithm in $[0, 1]$. We used 0.8 for the ratio of matching as suggested in the paper.
- **ImbML** [51] where the parameter m is tuned in $\{1, 10, 100, 1000, 10000\}$, the parameter λ in $\{0, 0.01, 0.1, 1, 10\}$ and the parameter a in $[0, 1]$. We also use a Bayesian optimization with 400 calls.

3.5.2 Results

The main results are reported in Table 3.1. Unsurprisingly, all metric learning methods perform better than a 3−**NN**. Furthermore, in terms of $F\text{-}Measure$, those which were designed to deal with imbalanced scenarios perform better than **LMNN**, **ITML** or **GMML**. However, the most competitive method is **MLFP**: the $F\text{-}Measure$ is increased on average by 1.4 points compared to the second best method (**ImbML**). More precisely, our **MLFP** outperforms all the other methods on 12 (over 28) datasets. The fact that **MLFP** works better than **ImbML** shows the advantage of learning a specific metric when computing distances to positive examples. So, it is not strictly necessary to take all terms of the $F\text{-}Measure$ into account in the loss function. Focusing on False Negative and False Positive Rates, as we propose here, is enough. Furthermore, as shown on Figure 3.3, both **ImbML** and **MLFP** focuses on the minority class, but they perform this task in a different way. Our method tries to reduce the number of FN by increasing the decision boundaries around each of positive example and then reduces the impact of each negatives by surrounding them. In **ImbML**, the possibility of having large margins in the learned space has the disadvantage of creating larger areas of negative classification and this potentially increases the risk of FN.

For the DGFiP datasets, the results are available in the Table 3.2. We can see that our method outperforms all other metric learning methods for these datasets. As our algorithm tends to focus on the positives, and therefore the fraudsters, it is obviously more suitable for them. This also comes from the fact that we know that, in this kind of context, fraudsters often try to mimic the behaviour of non-frauders in order to stay undetected. By artificially increasing the decision boundaries around known positives, it is easier to capture fraudsters in hiding.

[6] `https://scikit-optimize.github.io/`

Datasets	3−**NN**	**LMNN**	**ITML**	**GMML**	**IML**	**ImbML**	**MLFP**
Balance	0.880±0.018	0.874±0.019	0.931±0.032	0.888±0.025	0.886±0.029	**0.960±0.019**	0.874±0.010
Autompg	0.780±0.054	0.792±0.031	0.801±0.018	**0.823±0.034**	0.785±0.021	0.790±0.044	0.805±0.040
Ionosphere	0.745±0.015	0.803±0.049	0.831±0.054	0.764±0.056	0.823±0.044	0.786±0.053	**0.923±0.027**
Pima	0.601±0.042	0.591±0.037	0.583±0.022	0.579±0.035	0.591±0.037	0.575±0.026	**0.635±0.029**
Glass	0.735±0.049	0.710±0.064	**0.759±0.051**	0.750±0.032	0.710±0.064	0.716±0.043	0.747±0.042
German	0.407±0.049	0.358±0.029	0.430±0.073	0.407±0.030	0.352±0.029	0.388±0.043	**0.511±0.013**
Yeast1	0.511±0.022	0.493±0.035	0.508±0.014	0.498±0.022	0.506±0.033	0.510±0.009	**0.555±0.025**
Haberman	0.343±0.093	0.279±0.082	0.296±0.119	0.345±0.104	0.201±0.072	0.327±0.120	**0.428±0.042**
Vehicle3	0.504±0.049	0.533±0.032	0.569±0.024	0.562±0.024	0.527±0.031	**0.612±0.037**	0.562±0.045
Hayes	0.581±0.210	0.824±0.089	0.829±0.071	0.876±0.091	0.824±0.089	0.908±0.083	**0.930±0.106**
Segmentation	0.882±0.031	0.888±0.011	0.866±0.029	0.870±0.029	0.895±0.020	**0.909±0.028**	0.882±0.025
Abalone8	0.223±0.025	0.220±0.040	0.213±0.025	0.210±0.038	0.228±0.021	0.200±0.023	**0.336±0.016**
Yeast3	0.719±0.028	0.734±0.020	0.742±0.034	**0.747±0.031**	0.717±0.032	0.723±0.023	0.725±0.021
Ecoli3	0.487±0.177	0.541±0.169	0.501±0.220	**0.600±0.150**	0.552±0.201	0.541±0.182	0.550±0.118
Pageblocks	0.855±0.027	0.844±0.027	0.850±0.023	0.864±0.022	0.842±0.027	**0.865±0.021**	0.860±0.025
Satimage	0.688±0.034	0.707±0.038	0.710±0.024	0.682±0.028	0.710±0.039	**0.731±0.030**	0.697±0.024
Yeast-0-5-6-7-9vs4	0.364±0.112	0.415±0.184	0.494±0.156	0.383±0.066	0.326±0.144	0.473±0.126	**0.519±0.079**
Libras	0.694±0.188	0.725±0.105	0.722±0.204	0.667±0.272	0.690±0.120	**0.729±0.157**	0.694±0.189
Yeast-1vs7	0.455±0.174	**0.558±0.134**	0.411±0.262	0.402±0.138	**0.558±0.134**	0.498±0.152	0.531±0.089
Arrhythmia	0.178±0.151	**0.308±0.117**	0.238±0.217	0.124±0.152	0.216±0.171	0.306±0.208	0.119±0.023
Solar-flare-m0	0.112±0.009	0.141±0.094	0.088±0.044	0.119±0.074	0.110±0.136	0.095±0.008	**0.179±0.034**
Oil	0.412±0.061	0.612±0.125	0.602±0.083	0.413±0.128	0.631±0.069	**0.637±0.066**	0.554±0.040
Yeast4	0.250±0.101	0.187±0.108	0.341±0.062	0.327±0.137	0.196±0.098	0.371±0.118	**0.378±0.054**
Redwinequality4	0.062±0.075	0.057±0.114	0.027±0.053	0.055±0.068	0.000±0.000	0.031±0.062	**0.083±0.060**
Yeast5	0.612±0.038	**0.667±0.035**	0.662±0.084	0.588±0.065	0.659±0.029	0.614±0.096	0.605±0.032
Yeast6	0.560±0.205	0.578±0.246	0.523±0.205	0.458±0.307	**0.629±0.244**	0.606±0.148	0.564±0.193
Abalone17	0.000±0.000	0.000±0.000	0.029±0.057	0.000±0.000	0.000±0.000	**0.073±0.093**	0.053±0.047
Abalone20	0.000±0.000	0.000±0.000	0.000±0.000	0.000±0.000	0.044±0.089	0.000±0.000	**0.078±0.087**
Mean	0.487	0.516	0.520	0.500	0.507	0.535	**0.549**

Table 3.1: Mean results (and standard deviations) in terms of $F\text{-}Measure$ over 5 experiments for the different Metric Learning methods, with 3−**NN** as final classifier, on public datasets *sorted by imbalance ratio* (IR=m_-/m_+). The *mean* over all datasets among ML methods is given and the best results are in *bold*, the standard deviation is indicated with the ± sign.

Datasets	3−**NN**	**LMNN**	**ITML**	**GMML**	**IML**	**ImbML**	**MLFP**
DGFiP 9-2	0.231±0.047	0.266±0.063	0.232±0.102	0.186±0.070	0.306±0.053	0.187±0.057	**0.391±0.026**
DGFiP 4-2	0.049±0.059	0.076±0.097	0.114±0.097	0.136±0.025	0.099±0.089	0.071±0.094	**0.307±0.067**
DGFiP 8-1	0.167±0.047	0.164±0.041	0.102±0.048	0.131±0.070	0.203±0.047	0.137±0.058	**0.308±0.004**
DGFiP 8-2	0.175±0.044	0.161±0.064	0.132±0.067	0.168±0.021	0.129±0.031	0.161±0.047	**0.304±0.006**
DGFiP 9-1	0.067±0.087	0.064±0.053	0.140±0.059	0.103±0.092	0.150±0.096	0.184±0.109	**0.290±0.027**
DGFiP 4-1	0.094±0.078	0.029±0.057	0.000±0.000	0.055±0.068	0.000±0.000	0.033±0.067	**0.262±0.025**
DGFiP 16-1	0.058±0.048	0.139±0.123	0.065±0.080	0.159±0.105	**0.282±0.076**	0.119±0.102	0.207±0.016
DGFiP 16-2	0.076±0.039	0.035±0.044	0.078±0.040	0.164±0.086	0.107±0.070	0.164±0.164	**0.206±0.015**
Mean	0.115	0.117	0.108	0.138	0.160	0.132	**0.284**

Table 3.2: Mean results (and standard deviations) in terms of $F\text{-}Measure$ over 5 experiments for the different Metric Learning methods, with 3−**NN** as final classifier, on private datasets *sorted by imbalance ratio* (IR=m_-/m_+). The *mean* over all datasets among ML methods is given and the best results are in *bold*, the standard deviation is indicated with the ± sign.

In the theoretical part of this chapter, we have proved that learning a matrix $\mathbf{M}$ under the constraint $\lambda_{\max}(\mathbf{M}) \leq 1$ allows our algorithm to focus first on reducing FN. An illustration of the impact of this contraint in terms of False Negatives is shown in Figure 3.4 on the 28 datasets. This figure reports the percentage of false negatives **at test time** generated by the 3−**NN** algorithm and **MLFP** with or without the constraint. The results show that, compared to a 3−**NN** algorithm, **MLFP** systematically reduces the number of false negatives and thus has the desired effect. When comparing **MLFP** with and without the constraint, we can note that on 19 datasets out of 28, the use of the constraint $\lambda_{\max}(\mathbf{M}) \leq 1$ leads at test time to a smaller number of false negatives.

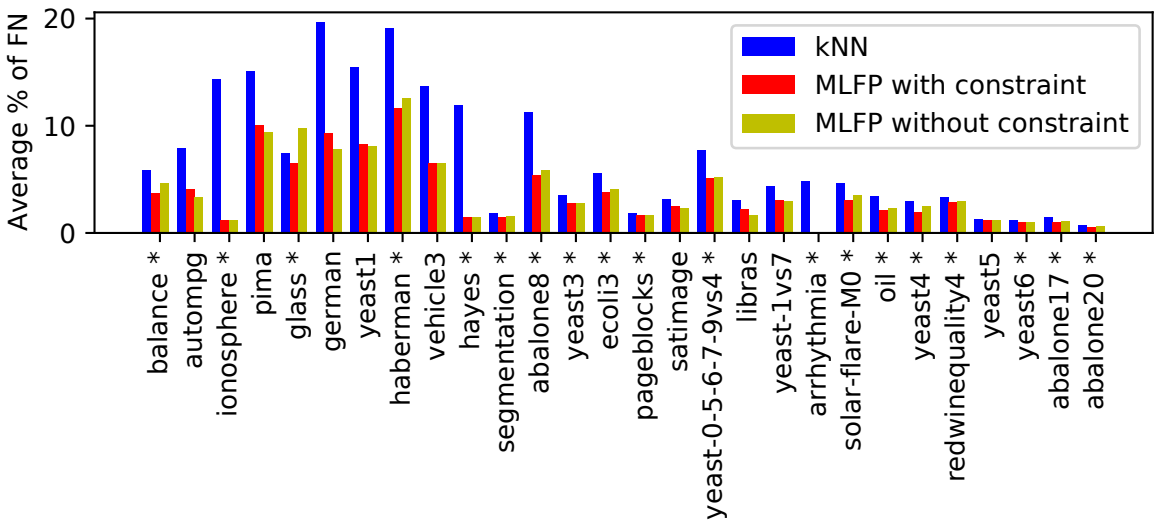

Figure 3.4: Average percentage of FN for each dataset at test time (see Section 3.5 for more details), for $k-$**NN** and **MLFP** with or without the constraint on $\lambda_{\max}$. On 19 datasets (with $*$) over 28, the number of FN is lower for the version with the constraint. Note that the number of FN is always lower with **MLFP** compared to $k-$**NN**.

3.6 Conclusion

In this chapter, we have proposed a new metric learning algorithm to deal with imbalanced datasets. In this setting, finding the good compromise between the false negative and false positive rates is still an open problem. The original contribution of this chapter comes from the optimization in our algorithm **MLFP** of a Mahalanobis distance which is *only* used to compare a new query to positive examples, while the Euclidean distance is still used when for comparing that query to negative samples. A constraint on the maximum eigenvalue of the learned matrix is introduced and has been shown to be provably efficient to reduce the false negative rate. Our chapter is supported by a theoretical study and an extensive experimental evaluation showing that **MLFP** outperforms state-of-the-art metric-learning methods.

This work opens the door to two promising lines of research. First, in **MLFP** we learn a linear projection of the data. One interesting perspective would consist in kernelizing our metric learning algorithm or designing a deep learning version allowing us to capture non linearity. A simpler solution might also consist in learning different local metrics for different regions of the input space as done in [130]. Second, as initiated in [102], combining a Mahalanobis distance with a sampling strategy might lead to a new family of imbalanced learning methods.

Datasets	1−**NN**	**LMNN**	**ITML**	**GMML**	**IML**	**ImbML**	**MLFP**
balance	0.855±0.028	0.862±0.033	0.907±0.049	0.857±0.026	0.856±0.032	**0.924±0.018**	0.850±0.024
autompg	0.751±0.072	0.765±0.056	0.794±0.029	**0.797±0.049**	0.775±0.073	0.795±0.034	0.795±0.037
ionosphere	0.796±0.032	0.741±0.101	0.790±0.049	0.839±0.047	0.796±0.042	0.849±0.052	**0.899±0.053**
pima	0.568±0.052	0.579±0.040	0.557±0.043	0.536±0.044	0.547±0.055	0.566±0.053	**0.595±0.043**
glass	0.753±0.053	0.698±0.114	**0.755±0.063**	0.732±0.059	0.750±0.048	0.736±0.058	0.743±0.060
german	0.382±0.023	0.422±0.023	0.403±0.043	0.368±0.030	0.444±0.027	0.418±0.046	**0.509±0.032**
yeast1	0.516±0.027	0.497±0.033	0.499±0.029	0.506±0.018	0.472±0.022	0.499±0.031	**0.525±0.024**
haberman	0.318±0.024	0.274±0.081	0.295±0.109	0.337±0.062	0.293±0.114	0.261±0.084	**0.421±0.039**
vehicle3	0.500±0.054	0.551±0.053	0.536±0.076	0.600±0.038	0.599±0.023	**0.612±0.027**	0.552±0.040
hayes	0.808±0.079	0.782±0.175	**0.850±0.050**	0.802±0.078	0.693±0.219	0.787±0.062	0.820±0.095
segmentation	0.898±0.029	0.914±0.016	0.905±0.029	0.906±0.032	0.906±0.024	**0.922±0.021**	0.896±0.031
abalone8	0.223±0.027	0.240±0.022	0.213±0.020	0.218±0.020	0.223±0.049	0.217±0.036	**0.295±0.017**
yeast3	0.662±0.022	0.643±0.026	0.654±0.034	0.679±0.028	0.650±0.021	**0.680±0.009**	0.664±0.018
ecoli3	0.548±0.129	0.456±0.141	0.420±0.253	0.452±0.117	0.475±0.091	0.552±0.101	**0.563±0.083**
pageblocks	**0.850±0.012**	0.845±0.014	0.840±0.027	0.842±0.018	0.842±0.012	0.845±0.015	0.847±0.013
satimage	0.697±0.037	0.703±0.043	0.693±0.030	0.685±0.032	0.681±0.046	**0.725±0.036**	0.697±0.033
Yeast-0-5-6-7-9vs4	0.369±0.027	0.383±0.133	**0.427±0.138**	0.367±0.075	0.362±0.090	0.385±0.112	0.377±0.047
libras	0.706±0.205	0.716±0.208	0.706±0.205	0.750±0.215	0.624±0.165	0.733±0.218	**0.756±0.237**
yeast-1vs7	0.349±0.122	0.352±0.142	**0.485±0.064**	0.415±0.091	0.216±0.140	0.469±0.107	0.428±0.134
arrhythmia	0.236±0.114	**0.330±0.220**	0.216±0.140	0.228±0.145	0.232±0.188	0.160±0.233	0.223±0.139
solar-flare-m0	0.183±0.075	0.181±0.051	0.184±0.089	**0.214±0.048**		0.160±0.052	0.179±0.025
oil	0.521±0.103	**0.705±0.097**	0.574±0.105	0.480±0.147	0.518±0.092	0.608±0.112	0.527±0.127
yeast4	0.287±0.091	0.297±0.057	0.246±0.060	**0.348±0.096**	0.335±0.089	0.303±0.078	0.253±0.046
redwinequality4	0.096±0.086	0.095±0.120	0.040±0.080	0.042±0.052	**0.114±0.127**	0.097±0.124	0.113±0.027
yeast5	0.660±0.160	0.636±0.155	0.634±0.122	0.667±0.152	0.650±0.119	**0.688±0.104**	0.678±0.136
yeast6	0.421±0.162	0.480±0.114	0.433±0.166	0.475±0.172	0.378±0.113	**0.481±0.091**	0.421±0.162
abalone17	0.124±0.074	**0.159±0.110**	0.055±0.049	0.156±0.115	0.076±0.042	0.117±0.067	**0.159±0.068**
abalone20	0.000±0.000	0.000±0.000	0.000±0.000	0.000±0.000	0.000±0.000	**0.095±0.124**	0.029±0.026
Mean	0.503	0.511	0.504	0.511	0.500	0.524	**0.529**

Table 3.3: Mean results (and standard deviations) in terms of *F-Measure* over 5 experiments for the different Metric Learning methods, with 1−**NN** as final classifier, on public datasets *sorted by imbalance ratio* (IR=m_-/m_+). The *mean* over all datasets among ML methods is given and the best results are in *bold*, the standard deviation is indicated with the ± sign.

Datasets	1−**NN**	**LMNN**	**ITML**	**GMML**	**IML**	**ImbML**	**MLFP**
DGFiP 9-2	0.308±0.081	0.283±0.084	0.281±0.072	0.251±0.124	0.263±0.105	0.278±0.053	**0.407±0.026**
DGFiP 4-2	0.183±0.109	0.210±0.085	0.309±0.083	0.131±0.111	0.316±0.083	0.176±0.066	**0.325±0.051**
DGFiP 8-1	0.153±0.064	0.144±0.037	0.171±0.034	0.148±0.059	0.201±0.086	0.165±0.025	**0.308±0.006**
DGFiP 8-2	0.236±0.068	0.196±0.045	0.156±0.050	0.233±0.042	0.229±0.052	0.215±0.079	**0.286±0.014**
DGFiP 9-1	0.178±0.128	0.112±0.065	0.131±0.074	0.194±0.109	0.119±0.069	0.146±0.055	**0.319±0.040**
DGFiP 4-1	0.177±0.098	0.111±0.166	0.183±0.113	0.075±0.093	0.134±0.069	0.104±0.093	**0.203±0.108**
DGFiP 16-1	0.152±0.094	0.152±0.063	0.189±0.071	0.160±0.108	0.170±0.050	0.126±0.079	**0.212±0.044**
DGFiP 16-2	0.142±0.033	0.159±0.104	0.121±0.043	0.181±0.036	0.075±0.006	0.165±0.059	**0.251±0.029**
Mean	0.191	0.171	0.193	0.172	0.188	0.172	**0.289**

Table 3.4: Mean results (and standard deviations) in terms of *F-Measure* over 5 experiments for the different Metric Learning methods, with 1−**NN** as final classifier, on private datasets *sorted by imbalance ratio* (IR=m_-/m_+). The *mean* over all datasets among ML methods is given and the best results are in *bold*, the standard deviation is indicated with the ± sign.

3.7 Proof of Theorems 2 and 3

Let us remind you the different notations. We will denote by $\mathbf{z} = (\mathbf{x}, y)$ the couple features-label where $\mathbf{x} \in \mathbb{R}^d$ and $y \in \{-1, 1\}$ and $S = \{\mathbf{z}_i\}_{i=1}^m$ a set of m training examples drawn from an unknown distribution $\mathcal{D}$. We denote by m_+ the number of positives and m_- the number of negatives. Thus the rate of positives α is equal to $\frac{m_+}{m}$.

Suppose that $\mathbf{x}'$ is a test instance, we also recall that:

- $\mathcal{D}_{\mathbf{M}} = \mathcal{D}_{\mathbf{M}}(\mathbf{x}', \mathbf{x}) = \sqrt{(\mathbf{x} - \mathbf{x}')^T \mathbf{M} (\mathbf{x} - \mathbf{x}')}$ if $\mathbf{x}$ is a positive instance,
- $\mathcal{D} = \mathcal{D}_{\mathbf{I}}(\mathbf{x}', \mathbf{x}) = \mathcal{D}(\mathbf{x}', \mathbf{x}) = \sqrt{(\mathbf{x} - \mathbf{x}')^T (\mathbf{x} - \mathbf{x}')}$ otherwise.

We are considering our optimization problem 3.1:

$$\min_{\mathbf{M} \in \mathbb{S}^+} \quad \frac{1}{m^3} \left((1-\alpha) \sum_{\substack{(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k) \\ y_i = y_j \neq y_k = -1}} \ell_{\mathrm{FN}}(\mathbf{M}, \mathbf{z}_i, \mathbf{z}_j, \mathbf{z}_k) + \alpha \sum_{\substack{(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k) \\ y_i = y_j \neq y_k = 1}} \ell_{\mathrm{FP}}(\mathbf{M}, \mathbf{z}_i, \mathbf{z}_j, \mathbf{z}_k) \right) + \mu \|\mathbf{M} - \mathbf{I}\|_{\mathcal{F}}^2.$$

Our loss function can thus be seen as :

$$\ell(\mathbf{M}, (z_i, z_j, z_k)) = \begin{cases} (1-\alpha) \times \ell_{\mathrm{FN}}(\mathbf{M}, \mathbf{z}_i, \mathbf{z}_j, \mathbf{z}_k) & \text{if } y_i = y_j = 1,\ y_k = -1, \\ \alpha \times \ell_{\mathrm{FP}}(\mathbf{M}, \mathbf{z}_i, \mathbf{z}_j, \mathbf{z}_k)] & \text{if } y_i = y_j = -1,\ y_k = 1, , \\ 0 & \text{otherwise,} \end{cases}$$

where ℓ_{FN} and ℓ_{FP} are defined by:

- $\ell_{\mathrm{FN}}(\mathbf{M}, \mathbf{z}_i, \mathbf{z}_j, \mathbf{z}_k) = [1 - c + \mathcal{D}_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j)^2 - \mathcal{D}(\mathbf{x}_i, \mathbf{x}_k)^2]_+$,
- $\ell_{\mathrm{FP}}(\mathbf{M}, \mathbf{z}_i, \mathbf{z}_j, \mathbf{z}_k) = [1 - c + \mathcal{D}(\mathbf{x}_i, \mathbf{x}_j)^2 - \mathcal{D}_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_k)^2]_+$.

In the following, we will also suppose that for all $\mathbf{x}$ we have: $\|\mathbf{x}\| \leq K$. Furthermore, we will denote by $\mathcal{R}_S$ and $\mathcal{R}$ respectively the empirical risk of ℓ over the training sample S and the true risk. More precisely, the empirical risk $\mathcal{R}_S$ is evaluated using a training set of size m which is used to build all the triplets and the true risk $\mathcal{R}$ is its expectation over all the samples of size m, i.e. $\mathcal{R} = \mathop{\mathbb{E}}_{S \sim \mathcal{D}^m} [\mathcal{R}_S]$.

In the following, we will also use the following constraint on $\mathbf{M}$:

$$\lambda_{\max}(\mathbf{M}) \leq 1, \text{ where } \lambda_{\max} \text{ is the largest eigenvalue of } \mathbf{M}.$$

Finally, due to the context of our study, i.e. imbalanced setting, $\alpha < 1/2$. Thus, $\alpha < 1 - \alpha$.

3.8 Generalization Guarantees

The aim of this section is to provide some generalization guarantees on our loss function according to the used loss function. Note that the following results give guarantees on the learned metric $\mathbf{M}$ which aims to find a good compromise between achieving a low False Negatives rate while keeping a reasonable False Positives rate.

3.8.1 Uniform Stability

In this section, we briefly restate the definition of stability and the generalization bound based on this notion.

Roughly speaking, an algorithm is *stable* if its output, in terms of difference between losses, does not change significantly under a small modification of the training sample. This variation must be bounded in $O(1/m)$ in terms of infinite norm where m is the size of the training set S *i.i.d.* from an unknown distribution $\mathcal{D}$.

Definition 6. *[Definition 6 [13]] A learning algorithm $\mathcal{A}$ has a uniform stability in $\frac{\kappa}{m}$ with respect to a loss function ℓ and parameter set θ, with κ a positive constant if:*

$$\forall S,\ \forall i,\ 1 \leq i \leq m,\ \sup_{Z} |\ell(\theta_S, Z) - \ell(\theta_{S^i}, Z)| \leq \frac{\kappa}{m},$$

where S is a learning sample of size m, $Z = (\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3) = ((\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_3, y_3))$ is a triplet of labeled examples, θ_S the model parameters learned from S, θ_{S^i} the model parameters learned from the sample S^i obtained by replacing the i^{th} example z_i from S by another example z'_i independent from S and drawn from $\mathcal{D}$. $\ell(\theta_S, \mathbf{x})$ is the loss suffered at $\mathbf{x}$.

In this definition, S^i represents the notion of small modification of the training sample. The following one aims to study the evolution of the loss function according to the label of the considered triplet.

Definition 7. *A loss function ℓ is said to be σ-admissible, with respect to the distance metric $\mathbf{M}$ if (i) it is convex with respect to its first argument and (ii) the following condition holds:*

$$\forall Z, Z' \quad \big|\ell(\mathbf{M}, Z) - \ell(\mathbf{M}, Z')\big| \leq \sigma,$$

where $Z = (z_i, z_j, z_k)$ and $Z' = (z'_i, z'_j, z'_k)$ are two triplets of examples.

3.8.2 Preliminary Results

We now introduce the results we need to derive our generalization guarantees:

Proposition 4. *Let $X_1, ..., X_m$ be m independent random variables taking values in $\mathbb{R}$ and let $U = f(X_1, ..., X_m)$. If, for each $1 \leq i \leq m$, there exists a constant c_i such that:*

$$\sup_{x_1,...,x_m \in \mathbb{R}} \big|f(x_1, ..., x_m) - f(x_1, ...x'_i, ..., x_m)\big| \leq c_i,$$

then, for any positive constant B, we have:

$$\mathbb{P}[|U - \mathbb{E}[U]| \geq B] \leq 2\exp\left(\frac{-2B^2}{\sum_{i=1}^m c_i^2}\right).$$

In the following, we set $D_S = \mathcal{R} - \mathcal{R}_S$. We then introduce the two following lemmas, for which the proof can be found in [10] (see the proofs of Lemma 8.9 and 8.10 respectively). However, note that results have been adapted to our context, i.e. for triplet based loss function. But the proofs can be easily adapted.

Lemma 1. *For any learning method of estimation error D_S and satisfying a uniform stability in $\frac{\kappa}{m}$, we have $\mathbb{E}_S[D_S] \leq \frac{2\kappa}{m}$.*

Lemma 2. *For any parameter matrix $\mathbf{M}$ using m training examples, and any loss function ℓ satisfying the σ-admissibility, we have the following bound:*

$$\forall i,\ 1 \leq i \leq m,\ |D_S - D_{S^i}| \leq \frac{2\kappa}{m} + \frac{2\sigma}{m}.$$

Using the above Proposition and the two Lemmas, we are able to get the following generalization bound:

Theorem 4. *Let $\delta > 0$ and $m > 1$. Let S be a sample of m randomly selected training examples and let $\mathbf{M}$ be the learned parameter matrix from an algorithm with uniform stability $\frac{\kappa}{m}$. Assuming that the loss function ℓ is k-Lipschitz and σ-admissible and let us denote by $\mathcal{R}_S$ its empirical risk.*
With probability $1-\delta$, we have the following bound on the true risk $\mathcal{R}$ of our loss function ℓ:

$$\mathcal{R} \leq \mathcal{R}_S + 2\frac{\kappa}{m} + (2\kappa + 2\sigma)\sqrt{\frac{\ln(2/\delta)}{2m}}.$$

3.8.3 Generalization Bound

We first prove that our function is k-Lipschitz according to the following definition.

Definition 8. *A loss function ℓ is k-Lipschitz with respect to its first argument if for any parameters matrices $\mathbf{M}$ and $\mathbf{M}'$, and for any triplets of labeled examples $Z = (\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3)$, we have:*

$$\left|\ell(\mathbf{M}, Z) - \ell(\mathbf{M}', Z)\right| \leq k \left|\mathbf{M} - \mathbf{M}'\right|_{\mathcal{F}}.$$

Lemma 3. *We now show that our loss function ℓ is k-Lipschitz with $k = 4(1-\alpha)K^2$*

Proof. We need to study two cases, according to the label of the triplets.

Case 1: $y_i = y_j = 1,\ y_k = -1$

$$\begin{aligned}
\left|\ell(\mathbf{M}, Z) - \ell(\mathbf{M}', Z)\right| &= (1-\alpha)\left|[1 - c + \mathcal{D}_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j)^2 - \mathcal{D}(\mathbf{x}_i, \mathbf{x}_k)^2]_+ \right. \\
&\qquad \left. -[1 - c + \mathcal{D}_{\mathbf{M}'}(\mathbf{x}_i, \mathbf{x}_j)^2 - \mathcal{D}(\mathbf{x}_i, \mathbf{x}_k)^2]_+\right|, \\
&\leq (1-\alpha)\left|\mathcal{D}_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j)^2 - \mathcal{D}_{\mathbf{M}'}(\mathbf{x}_i, \mathbf{x}_j)^2\right|, \\
&= (1-\alpha)\left|(\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{M} - \mathbf{M}')(\mathbf{x}_i - \mathbf{x}_j)\right|, \\
&= (1-\alpha)\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 \|\mathbf{M} - \mathbf{M}'\|_{\mathcal{F}}, \\
\left|\ell(\mathbf{M}, Z) - \ell(\mathbf{M}', Z)\right| &\leq 4(1-\alpha)K^2\|\mathbf{M} - \mathbf{M}'\|_{\mathcal{F}}
\end{aligned}$$

where the second line uses the fact that the hinge loss is 1-Lipschitz, the third line uses the linearity of the difference with respect to $\mathbf{M}, \mathbf{M}'$, the fourth line uses usual properties on norms and the last line the fact that $\|\mathbf{x}\| \leq K$.

Case 2: $y_i = y_j = -1,\ y_k = 1$

The proof is similar to the proof given in the previous case and leads to the following result:

$$\left|\ell(\mathbf{M}, Z) - \ell(\mathbf{M}', Z)\right| \leq 4\alpha K^2\|\mathbf{M} - \mathbf{M}'\|_{\mathcal{F}}.$$

We conclude by taking the maximum of the two previous values. Thus $k = 4(1-\alpha)K^2$. □

Now, we have to prove that our loss function is σ-admissible according to the definition 7.

Lemma 4. *The loss function ℓ defined by (3.1) is σ-admissible with respect to the distance metric $\mathbf{M}$, with $\sigma = (1-\alpha)(1 - c + 4K^2)$.*

Proof. Needless to say that the loss function ℓ is convex with respect to $\mathbf{M}$ as the sum of two convex functions. Indeed, both of them are linear *w.r.t.* $\mathbf{M}$ and the maximum of two convex functions remains convex.

Furthermore, because our loss function can be equal to zero for some labels of our triplets, we are looking for the greatest value than our loss function ℓ can achieve.

Using our previous result, we can bound the first part ℓ_{FN} by: $(1-\alpha)(1-c+4K^2)$ and the last term ℓ_{FP} by: $\alpha(1-c+4K^2)$.

Finally:

$$\forall\ Z, Z'\ \left|\ell(\mathbf{M}, Z) - \ell(\mathbf{M}, Z')\right| \leq \max\left((1-\alpha)(1-c+4K^2), \alpha(1-c+4K^2)\right).$$

Thus, $\sigma = (1-\alpha)(1-c+4K^2)$. □

Definition 9. *A learning algorithm has a uniform stability in $\frac{\kappa}{m}$ where κ is a positive constant, if given any training set S we have:*

$$\forall i,\ \sup_{Z} |\ell(\mathbf{M}, Z) - \ell(\mathbf{M}^i, Z)| \leq \frac{\kappa}{m},$$

where M^i is the matrix learned with a training set S^i which differs from S of only one example $(\mathbf{x}_i \to \mathbf{x}'_i)$.

For the sake of clarity for the following development, let us denote by F_S the objective function to optimize over the training set S, i.e. $F_S = \frac{1}{m^3}\sum_{\mathbf{x}_i,\mathbf{x}_j,\mathbf{x}_k} \ell(\mathbf{M}, Z) + \mu\|\mathbf{M}-\mathbf{I}\|^2_{\mathcal{F}}$. To compute the constant of uniform stability, we first need the following technical lemma:

Lemma 5. *Let S be a learning sample, let F_S and F_{S^i} be two objective functions with respect to two samples S and S^i and let $\mathbf{M}$ and $\mathbf{M}^i$ be their respective minimizers. We also define $\Delta\mathbf{M} = \mathbf{M}^i - \mathbf{M}$ and recall that $N(\mathbf{M}) = \mu\|\mathbf{M}-\mathbf{I}\|^2_{\mathcal{F}}$. For all $t \in [0,1]$, we have:*

$$N(\mathbf{M}) - N(\mathbf{M}+t\Delta\mathbf{M}) + N(\mathbf{M}^i) - N(\mathbf{M}^i - t\Delta\mathbf{M}) \leq \frac{2t}{\mu m^3}[3m(m-1)+1] \times \left(4(1-\alpha)K^2\right) \times \|\Delta\mathbf{M}\|_{\mathcal{F}}.$$

Proof. Since ℓ (the hinge loss) is convex, so is the empirical risk and thus for all $t \in [0,1]$ we have the two following inequalities:

$$\mathcal{R}_{S^i}(\mathbf{M}+t\Delta\mathbf{M}) - \mathcal{R}_{S^i}(\mathbf{M}, R) \leq t\mathcal{R}_{S^i}(\mathbf{M}^i) - t\mathcal{R}_{S^i}(\mathbf{M}).$$

and

$$\mathcal{R}_{S^i}(\mathbf{M}^i - t\Delta\mathbf{M}) - \mathcal{R}_{S^i}(\mathbf{M}^i) \leq t\mathcal{R}_{S^i}(\mathbf{M}) - t\mathcal{R}_{S^i}(\mathbf{M}^i).$$

We get the second inequality by swapping the role of $\mathbf{M}$ and $\mathbf{M}^i$. If we sum these two inequalities, the right hand side vanishes and we obtain:

$$\mathcal{R}_{S^i}(\mathbf{M}+t\Delta\mathbf{M}) - \mathcal{R}_{S^i}(\mathbf{M}) + \mathcal{R}_{S^i}(\mathbf{M}^i - t\Delta\mathbf{M}) - \mathcal{R}_{S^i}(\mathbf{M}^i) \leq 0. \tag{3.6}$$

By assumption on $\mathbf{M}$ and $\mathbf{M}^i$ we have:

$$\begin{aligned} F_S(\mathbf{M}) - F_S(\mathbf{M}+t\Delta\mathbf{M}) &\leq 0, \\ F_{S^i}(\mathbf{M}^i) - F_{S^i}(\mathbf{M}^i - t\Delta\mathbf{M}) &\leq 0, \end{aligned}$$

then, summing the two previous inequalities and using (3.6), we get:

$$\begin{aligned} &\mathcal{R}_{S^i}(\mathbf{M}+t\Delta\mathbf{M}) - \mathcal{R}_S(\mathbf{M}+t\Delta\mathbf{M}) - \mathcal{R}_{S^i}(\mathbf{M}) + \mathcal{R}_S(\mathbf{M}) \\ &\quad + \mu[\|\mathbf{M}-\mathbf{I}\|^2_F + \|\mathbf{M}^i-\mathbf{I}\|^2_F - \|\mathbf{M}+t\Delta\mathbf{M}-\mathbf{I}\|^2_F - \|\mathbf{M}^i - t\Delta\mathbf{M}-\mathbf{I}\|^2_F] \leq 0. \end{aligned} \tag{3.7}$$

We now focus on the first part of the previous inequality. For the sake of simplicity, let us set:

$$\begin{aligned} H &= \mathcal{R}_S(\mathbf{M}+t\Delta\mathbf{M}) - \mathcal{R}_{S^i}(\mathbf{M}+t\Delta\mathbf{M}) + \mathcal{R}_{S^i}(\mathbf{M}) - \mathcal{R}_S(\mathbf{M}) \\ H &\leq |\mathcal{R}_S(\mathbf{M}+t\Delta\mathbf{M}) - \mathcal{R}_{S^i}(\mathbf{M}+t\Delta\mathbf{M}) + \mathcal{R}_{S^i}(\mathbf{M}) - \mathcal{R}_S(\mathbf{M})| \\ &\leq \frac{1}{m^3}\Bigg| \sum_{z_i,z_j,z_k \in S^l} \ell(\mathbf{M}, z_i^l, z_j^l, z_k^l) - \sum_{z_i,z_j,z_k \in S} \ell(\mathbf{M}, z_i, z_j, z_k) \\ &\quad + \sum_{z_i,z_j,z_k \in S} \ell(\mathbf{M}+t\Delta\mathbf{M}, z_i, z_j, z_k) - \sum_{z_i,z_j,z_k \in S^l} \ell(\mathbf{M}+t\Delta\mathbf{M}, z_i^l, z_j^l, z_k^l)\Bigg| \end{aligned}$$

where S and S^l differ from the l-th example, i.e. $\forall i,j,k \neq l,\ z_i = z_i^l,\ z_j = z_j^l$ and $z_k = z_k^l$.

We will now focus on the first difference in the previous expression, i.e. on:

$$\sum_{z_i,z_j,z_k \in S^l} \ell(\mathbf{M}, z_i^l, z_j^l, z_k^l) - \sum_{z_i,z_j,z_k \in S} \ell(\mathbf{M}, z_i, z_j, z_k).$$

This difference can be decomposed into two parts according to the value of the index i: when $i = l$ and when $i \neq l$:

$$\begin{aligned} &\sum_{j=1}^{m}\sum_{k=1}^{m}\Big(\ell(\mathbf{M}, z_l^l, z_j^l, z_k^l) - \ell(\mathbf{M}, z_l, z_j, z_k)\Big) \\ &+\sum_{i\neq l}^{m}\sum_{j=1}^{m}\sum_{k=1}^{m}\Big(\ell(\mathbf{M}, z_i^l, z_j^l, z_k^l) - \ell(\mathbf{M}, z_i, z_j, z_k)\Big) \end{aligned}$$

The first part of the decomposition is composed of m^2 terms that are at least not equal to zero. We, thus have to work on the second part of the decomposition has it contains some terms that are equal to zero. We will have to do this process two times as follows:

$$
\begin{aligned}
& \sum_{j=1}^{m}\sum_{k=1}^{m}\Big(\ell(\mathbf{M}, z_l^l, z_j^l, z_k^l) - \ell(\mathbf{M}, z_l, z_j, z_k)\Big) \\
& + \sum_{i\neq l}^{m}\sum_{j=1}^{m}\sum_{k=1}^{m}\Big(\ell(\mathbf{M}, z_i^l, z_j^l, z_k^l) - \ell(\mathbf{M}, z_i, z_j, z_k)\Big), \\
= & \sum_{j=1}^{m}\sum_{k=1}^{m}\Big(\ell(\mathbf{M}, z_l^l, z_j^l, z_k^l) - \ell(\mathbf{M}, z_l, z_j, z_k)\Big) \\
& + \sum_{i\neq l}^{m}\sum_{k=1}^{m}\Big(\ell(\mathbf{M}, z_i^l, z_l^l, z_k^l) - \ell(\mathbf{M}, z_i, z_l, z_k)\Big) \\
& + \sum_{i\neq l}^{m}\sum_{j\neq l}^{m}\sum_{k=1}^{m}\Big(\ell(\mathbf{M}, z_i^l, z_j^l, z_k^l) - \ell(\mathbf{M}, z_i, z_j, z_k)\Big), \\
= & \sum_{j=1}^{m}\sum_{k=1}^{m}\Big(\ell(\mathbf{M}, z_l^l, z_j^l, z_k^l) - \ell(\mathbf{M}, z_l, z_j, z_k)\Big) \\
& + \sum_{i\neq l}^{m}\sum_{k=1}^{m}\Big(\ell(\mathbf{M}, z_i^l, z_l^l, z_k^l) - \ell(\mathbf{M}, z_i, z_l, z_k)\Big) \\
& + \sum_{i\neq l}^{m}\sum_{j\neq l}^{m}\Big(\ell(\mathbf{M}, z_i^l, z_j^l, z_l^l) - \ell(\mathbf{M}, z_i, z_j, z_l)\Big) \\
& + \sum_{i\neq l}^{m}\sum_{j\neq l}^{m}\sum_{k\neq l}^{m}\underbrace{\Big(\ell(\mathbf{M}, z_i^l, z_j^l, z_k^l) - \ell(\mathbf{M}, z_i, z_j, z_k)\Big)}_{=0}.
\end{aligned}
$$

All these sums are respectively composed of m^2, $m(m-1)$ and $(m-1)^2$ terms and the last $(m-1)^3$ terms are all equal to zero. Furthermore: $m^2+m(m-1)+(m-1)^2 = 3m(m-1)+1$, so that we have to find a bound on the supremum of the difference:

$$[3m(m-1)+1]\sup_{Z,Z'}|\ell(\mathbf{M}, Z) - \ell(\mathbf{M}, Z') + \ell(\mathbf{M}+t\Delta\mathbf{M}, Z) - \ell(\mathbf{M}+t\Delta\mathbf{M}, Z')|.$$

Thus, H can be upper-bounded by:

$$H \leq \frac{1}{m^3}[3m(m-1)+1]\,(\sup_{Z,Z'}|\ell(\mathbf{M}, Z) - \ell(\mathbf{M}+, Z') + \ell(\mathbf{M}+t\Delta\mathbf{M}, Z) - \ell(\mathbf{M}+t\Delta\mathbf{M}, Z')|).$$

We can then write:

$$
\begin{aligned}
H &\leq \frac{1}{m^3}[3m(m-1)+1]\,(\sup_{Z}|\ell(\mathbf{M}+t\Delta\mathbf{M}, Z) - \ell(\mathbf{M}, Z)| \\
&\qquad + \sup_{Z'}|\ell(\mathbf{M}+t\Delta\mathbf{M}, Z') - \ell(\mathbf{M}+, Z')|), \\
&\leq \frac{2t}{m^3}[3m(m-1)+1]\times\|\Delta\mathbf{M}\|_{\mathcal{F}}\times\left(4(1-\alpha)K^2\right),
\end{aligned}
$$

where the last lines uses Lemma 3 and properties on norms. Finally, we have :

$$N(\mathbf{M})-N(\mathbf{M}+t\Delta\mathbf{M})+N(\mathbf{M}^i)-N(\mathbf{M}^i-t\Delta\mathbf{M}) \leq \frac{2t}{\mu m^3}[3m(m-1)+1]\times\left(4(1-\alpha)K^2\right)\times\|\Delta\mathbf{M}\|_{\mathcal{F}} \tag{3.8}$$

□

We are now able to prove the uniform stability of our algorithm.

Theorem 5. *Let S be a learning sample of size m, the algorithm (3.1) has a uniform stability in $\frac{\kappa}{m}$ with $\kappa = \frac{6}{\mu} \times \left(4(1-\alpha)K^2\right)^2$.*

Proof. Let us set $t = \frac{1}{2}$ in the result of Lemma 5 and we focus on the left hand side of this result. We have:

$$\begin{aligned} f(\mathbf{M}) &= \|\mathbf{M}-\mathbf{I}\|_F^2 + \|\mathbf{M}^i-\mathbf{I}\|_F^2 - \|\frac{1}{2}(\mathbf{M}+\mathbf{M}^i)-\mathbf{I}\|_F^2 - \|\frac{1}{2}(\mathbf{M}+\mathbf{M}^i)-\mathbf{I}\|_F^2, \\ &= \|\mathbf{M}-\mathbf{I}\|_F^2 + \|\mathbf{M}^i-\mathbf{I}\|_F^2 - \frac{1}{2}\|\mathbf{M}+\mathbf{M}^i-\mathbf{I}\|_F^2, \\ f(\mathbf{M}) &= \frac{1}{2}\|\mathbf{M}-\mathbf{M}^i\|_F^2. \end{aligned}$$

Then, using Lemma 5, we get the following bound on $\|\Delta\mathbf{M}\|_{\mathcal{F}}$.

$$\begin{aligned} \|\Delta\mathbf{M}\|_{\mathcal{F}}^2 &\leq \frac{8}{\mu m^3}[3m(m-1)+1] \times \left((1-\alpha)K^2\right) \times \|\Delta\mathbf{M}\|_{\mathcal{F}}, \\ \|\Delta\mathbf{M}\|_{\mathcal{F}} &\leq \frac{8}{\mu m^3}[3m(m-1)+1] \times \left((1-\alpha)K^2\right). \end{aligned}$$

To prove the uniform stability of our algorithm, it remains to find the value κ such that:

$$\forall S,\ \forall i,\ 1 \leq i \leq m,\ \sup_Z |\ell(\mathbf{M},Z) - \ell(\mathbf{M}^i,Z)| \leq \frac{\kappa}{m}.$$

To do this, we use the fact that our loss function ℓ is k-Lipsichtz with $k = \left(4(1-\alpha)K^2\right)$ and our upper-bound on $\|\Delta\mathbf{M}\|_{\mathcal{F}}$. It gives:

$$\begin{aligned} |\ell(\mathbf{M},Z) - \ell(\mathbf{M}^i,Z)| &\leq k\|\Delta\mathbf{M}\|_{\mathcal{F}}, \\ &\leq \frac{2k^2(3m^2-3m+1)}{\mu m^3}. \end{aligned}$$

Finally:

$$\forall S,\ \forall i,\ 1 \leq i \leq m,\ \sup_Z |\ell(\mathbf{M},Z) - \ell(\mathbf{M}^i,Z)| \leq \frac{\kappa}{m^3},$$

with $\kappa = \frac{4(3m^2-3m+1)}{\mu} \times \left((1-\alpha)K^2\right)^2$. □

For the sake of simplicity, we will simplify this result in the following. Note that for all $m \geq 1$, $\frac{3m^2-3m+1}{m^3} \leq \frac{3}{m}$. Thus, our algorithm has a uniform stability in $\frac{\kappa}{m}$ with $\kappa = \frac{12}{\mu} \times \left((1-\alpha)K^2\right)^2$.

We can now apply Theorem 4 to our algorithm and get the following result:

Theorem 6. *Let $\delta > 0$ and $m > 1$. With probability $1-\delta$, we have the following bound on the true risk $\mathcal{R}$ of our loss function ℓ:*

$$\mathcal{R} \leq \mathcal{R}_S + 2\frac{\kappa}{m} + (2\kappa + 2\sigma)\sqrt{\frac{\ln(2/\delta)}{2m}},$$

with:

$$\kappa = \frac{12}{\mu} \times \left((1-\alpha)K^2\right)^2.$$

and

$$\sigma = (1-\alpha)(1-c+4K^2).$$

Proof. The proof is consequence of Theorem 4 and Lemma 4. □

3.9 Classification Guarantees - Proof

We now give a proof of the Theorem 3 provided in the Section 3.4.2.

Proof. We first begin with the FP rate. We can note that the hinge loss can be a surrogate for the indicator function as follows:

$$\mathbb{1}_{\{d_{\mathbf{M}}(\mathbf{x},\mathbf{x}_{p_i})\leq d(\mathbf{x},\mathbf{x}_n)\}} = \mathbb{1}_{\{d_{\mathbf{M}}(\mathbf{x},\mathbf{x}_{p_i})^2\leq d(\mathbf{x},\mathbf{x}_n)^2\}} \leq \left[1 + d(\mathbf{x},\mathbf{x}_n)^2 - d_{\mathbf{M}}(\mathbf{x}_i,\mathbf{x}_p)^2\right]_+,$$

We can recognize one of the term of our optimization Problem (3.1) with the hyper-parameter $c = 0$.

We recall that each labeled example is denoted as $\mathbf{z} = (\mathbf{x}, y)$. Then, we have:

$$\begin{aligned}
\mathcal{R}_{FP} &\leq \mathbb{E}_{S\sim D^m}\mathbb{E}_{\mathbf{z}\sim D}\left[1 + d(\mathbf{x},\mathbf{x}_n)^2 - d_{\mathbf{M}}(\mathbf{x},\mathbf{x}_p)^2\right]_+ \times \mathbb{1}_{\{y=-1\}} \\
&\leq \mathbb{E}_{S'\sim D^{m+1}}\mathbb{E}_{\mathbf{z}_i,\mathbf{z}_j,\mathbf{z}_k\in S'}\left[1 + d(\mathbf{x}_i,\mathbf{x}_j)^2 - d_{\mathbf{M}}(\mathbf{x}_i,\mathbf{x}_k)^2\right]_+, \times \mathbb{1}_{\{y_i=y_j=-1\neq y_k\}} \\
&\leq \mathbb{E}_{S'\sim D^{m+1}}\mathbb{E}_{\mathbf{z}_i,\mathbf{z}_j,\mathbf{z}_k\in S'}\left[\frac{\alpha}{\alpha}\left[1 + d(\mathbf{x}_i,\mathbf{x}_j)^2 - d_{\mathbf{M}}(\mathbf{x}_i,\mathbf{x}_k)^2\right]_+ \times \mathbb{1}_{\{y_i=y_j=-1\neq y_k\}} + \right. \\
&\quad \left. \frac{1-\alpha}{\alpha}\left(\left[1 + d_{\mathbf{M}}(\mathbf{x}_i,\mathbf{x}_j)^2 - d(\mathbf{x}_i,\mathbf{x}_k)^2\right]_+ \times \mathbb{1}_{\{y_i=y_j=1\neq y_k\}}\right)\right], \\
&\leq \mathbb{E}_{S'\sim D^{m+1}}\mathbb{E}_{\mathbf{z}_i,\mathbf{z}_j,\mathbf{z}_k\in S'}\left[\frac{1}{\alpha}\left(\alpha\left[1 + d(\mathbf{x}_i,\mathbf{x}_j)^2 - d_{\mathbf{M}}(\mathbf{x}_i,\mathbf{x}_k)^2\right]_+ \times \mathbb{1}_{\{y_i=y_j=-1\neq y_k\}} + \right.\right. \\
&\quad \left.\left. (1-\alpha)\left[1 + d_{\mathbf{M}}(\mathbf{x}_i,\mathbf{x}_j)^2 - d(\mathbf{x}_i,\mathbf{x}_k)^2\right]_+ \times \mathbb{1}_{\{y_i=y_j=1\neq y_k\}}\right)\right], \\
&\leq \frac{1}{\alpha}\mathcal{R}.
\end{aligned}$$

The second inequality is obtained by the *i.i.d.* aspect of the expectation. The third inequality is due to the fact that the second term in the sum is positive. Finally, one can note that the right-hand side of the last inequality corresponds to a weighted version of the true risk with respect to the loss used in Problem (3.1) with $c = 0$ and where we take an expectation over all the samples of size $m + 1$.

The result is obtained by combining the results of Theorems 2 and 6 over the true risk defined above.

The bound for the FN rate can be obtained in a similar way. Using the same arguments, one can show that:

$$\mathcal{R}_{FN} \leq \frac{1}{1-\alpha}\mathcal{R}.$$

Applying Theorems 2 and 6 to the above risk leads to the result. □

Chapter 4

Tree-Based Ranking for Interpretable Fraud Detection

This chapter is based on the following publications

Rémi Viola, Léo Gautheron, Amaury Habrard and Marc Sebban. MetaAP: a Meta-Tree-Based Ranking Algorithm Optimizing the Average Precision From Imbalanced Data. In *Pattern Recognition Letters*, (revised version under review). 2022 [116].

Abstract

In this Chapter, we address the challenging problem of learning to rank from highly imbalanced data. This scenario requires to resort to specific metrics able to account the scarcity of positive examples, the data of interest that belongs to the minority class. For this purpose, we present **MetaAP**, a tree-based ranking algorithm, which induces meta-trees (in the form of trees of trees) by optimizing the *Average Precision* (*AP*). This latter has been shown in the literature to be more relevant than the area under the *ROC* curve (*AUC-ROC*) when the objective is to push positive examples at the very top of the list. This effect of the *AP* in tree-based ranking is particularly desired to address fraud detection tasks where (i) the budget is often constrained (in terms of number of agents *w.r.t.* the number of cases) and (ii) the interpretability of the induced models is required. After an extensive comparative study on many public datasets showing the ability of **MetaAP** to optimize the *AP* in highly imbalanced situations, we tackle again the tax fraud detection task coming from our partnership with the French Ministry of Economy and Finance. The results show that **MetaAP** is able to make the tax audit process significantly more efficient.

4.1 Introduction

Learning to rank from highly imbalanced datasets where positive examples are very scarce has received much attention during the past years from the Machine Learning community. Indeed, this challenging topic opened the door to many methodological questions: *Which loss function to optimize? Can we derive generalization guarantees in such a scenario? How to efficiently balance the datasets? Can we learn interpretable models? How to rank under budget constraints?* etc.

As seen before, fraud detection [2] falls into this scope of imbalanced learning to rank, where the number of fraudsters is small compared to the huge amount of normal cases (also called negative examples). It has become a key issue for e-commerce companies and government agencies which are facing a tremendous growth of the data collected that have to be processed by a relatively limited number of human controllers. Therefore, fraud detection is subject nowadays

to a compelling need for automatic interpretable systems for supporting human decision making. Unlike a standard anomaly detection task [4] where an abnormal data often takes the form of an outlier, the peculiarity of fraud detection is that fraudsters often aim to mimic a normal behavior that makes the identification much more challenging.

Even so, in fraud detection, we are often faced with the problem of learning from highly imbalanced data [57] because the number of positive examples, the frauds, is very small compared to the number of normal examples. As explained before in this thesis, one way to address this task is to resort to *sampling* strategies [23, 108, 124]. While oversampling/data augmentation techniques can be used to generate dummy data artificially like in SMOTE-based methods [23] or in adversarial approaches [33], undersampling aims at removing samples from the majority class as done with Tomek's Link [108] or in ENN [124]. Even though a sampling method has the indisputable advantage of (re)balancing the datasets and allowing then the use of classical learning algorithms, in highly imbalanced scenarios, these methods do not succeed in generating enough diversity for improving significantly the results compared to the required effort, as recently shown in [20]. Other techniques to deal with the class imbalance problem include *cost-sensitive methods* [36] which nevertheless require a difficult tuning of the miss-classification costs, *(deep) metric learning methods* [39, 119, 74, 121, 50] which often requires a large amount of data and/or a costly optimization process, or *boosting-based models* [25, 49, 43] which are not easily interpretable. It turns out that interpretability is sometimes a desired property in fraud detection. Indeed, the detected suspicious cases are typically sent as alerts to the control department according to their position in the ranking, *i.e.* their probability of being a fraud. These top-ranked cases that are judged as frauds by the automatic system are then meticulously checked by a human controller whose analysis needs to be guided by the criteria that led to this decision. It is therefore often crucial for the prediction model to be supported by explainable decisions. As mentioned in [94], it is key to create methods that are readily interpretable rather than creating black boxes that will have to be explained later, often in imprecise ways.

In this context, decision trees seem to provide a good trade-off between accuracy and interpretability, beyond their natural capacity to deal with both quantitative and qualitative features. However, in order to address the issues induced by imbalanced datasets, they have to be optimized according to criteria that are able to take into account the scarcity of the positive examples compared to the large amount of negative samples (*e.g.* non fraudulent cases). Moreover, in a learning to rank perspective, the decision tree should be learned as a scoring function projecting the data onto the real line. This is the goal of *TreeRank*, introduced in the seminal work of Clemençon and Vayatis [27] and exploited in several variants since then. *TreeRank* recursively maximizes the area under the *ROC* curve (*AUC-ROC*) allowing the induction of a tree that optimizes the probability to rank a positive example above a negative one. As illustrated in this Chapter, *TreeRank* seems to behave better when facing imbalanced datasets than standard decision trees induced by using the classic Gini or Entropy criteria for recursively splitting the nodes. However, as pointed out by [19, 43], in applications where only the very top rank will be used because of budget constraints, like in fraud detection where the number of human controllers is limited, the *AUC-ROC* does not seem to be the most suitable criterion. Indeed, a *AUC-ROC*-based algorithm will put a lot of effort into the enhancement of the scoring of currently poorly ranked samples at the expense of the positive data that are already favorably positioned in the ranking. To overcome this issue, Frery et al. [43] have shown that the *Average Precision* (*AP*) is a more adapted metric when we are mainly interested in the top of the list, even if it is at the price of definitely dropping out positive examples that are a bit further down in the ranking.

Inspired by *TreeRank*, we design in this Chapter a new algorithm, called **MetaAP**, which optimizes the *AP* by building a tree of local trees, referred to as meta-trees. A natural but sub-optimal method (as seen later in the experimental part) to address this task would consist in optimizing the hyperparameters of the trees (depth, splitting criterion, etc.) according to

the AP. The novelty of **MetaAP** comes from the direct optimization of this measure, viewed as a loss function, during the learning process. However, this task is hard because the AP takes the form of a non convex (even with well-made surrogates) and non separable function (*i.e.* the loss for one point depends on the others). The originality of the proposed approach consists in exploiting the slope coefficient of the tangents to the (1-Precision)-Recall curve (which plays a key role in the definition of AP) to order and merge the leaves of the local tree. In this way, **MetaAP** aims at recursively optimizing the AP and presents the valuable property of generating compact and interpretable models. Our method is supported by an extensive comparative study on 28 public datasets showing that the smaller the proportion of positives we learn from, the better **MetaAP** outperforms the competing algorithms. This comment is confirmed when the *Precision@k* (corresponding to the number of positives among the top k ordered samples) is used as the evaluation measure that shows that **MetaAP** is particularly adapted to applications where the number of possible controls is limited. We further analyze a random forest version of our method and compare it with classic random forests and gradient boosting. Finally, we study the behavior of **MetaAP** to address a tax fraud detection task coming from our partnership with the French Ministry of Economy and Finance. The results show that **MetaAP** is able to make the tax audit process more efficient.

The rest of this Chapter is organized as follows: Section 4.2 is devoted to the presentation of the notations and evaluation measures. Section 4.3 is dedicated to the state of the art. We introduce **MetaAP** in Section 4.4 and present the experiments in Section 4.5.

4.2 Notations and Evaluation Measures

Let us remind that we consider a binary supervised learning task, with a training set $S = \{z_i = (\mathbf{x}_i, y_i)\}_{i=1}^m$ composed of m labeled examples with $\mathbf{x}_i \in \mathcal{X} = \mathbb{R}^d$, a feature vector and $y_i \in \mathcal{Y} = \{-1; +1\}$, a label. When $y_i = 1$ (*resp.* $y_i = -1$), x_i is a positive (*resp.* negative) example belonging to the minority (*resp.* majority). S is supposed to be independently and identically drawn according to an unknown joint distribution $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$.

Considering that we aim at addressing the problem through the lens of a learning to rank model in the context of imbalanced datasets, the choice of the metric to be optimized is key. As already mentioned in Chapter 1, in such a setting, three measures are usually used: the *AUC-ROC*, the *Average Precision* (*AP*) and the *Precision@k*. We briefly recap these three concepts in the following.

The *ROC* curve is the representation of the True Positive Rate (TPR / *Recall*) versus the False Positive Rate (FPR / False alarm rate) at different thresholds. TPR measures the capacity of the model to retrieve positive examples while FPR corresponds to the proportion of false alarms (fraction of negatives predicted as positives). More formally,

$$TPR = Recall = \frac{TP}{TP + FN} \quad \text{and} \quad FPR = \frac{FP}{FP + TN}, \tag{4.1}$$

where FP (*resp.* FN) is the number of false positives (*resp.* false negatives) and TP (*resp.* TN) is the number of true positives (*resp.* true negatives). The *AUC-ROC* corresponds to the area under this *ROC*-curve and measures the probability of the model to rank a positive example above a negative one. It can be computed as follows:

$$AUC\text{-}ROC = \frac{1}{m^+ \times m^-} \sum_{i=1}^{m^+} \sum_{j=1}^{m^-} \mathbf{I}_{0.5}(f(x_i^+) - f(x_j^-)),$$

- m^+ (*resp.* m^-) is the number of positives (*resp.* negatives) in S,
- x_i^+ is the i^{th} positive sample; x_j^- is the j^{th} negative example,
- f is the scoring function assigning a probability to be positive,

- $\mathbf{I}_{0.5}$ is an indicator function equal to 1 when $f(x_i^+) - f(x_j^-) > 0$, $\frac{1}{2}$ when $f(x_i^+) - f(x_j^-) = 0$ and 0 otherwise.

Another important tool when working with imbalanced data is the *Precision-Recall* curve. It represents the *Precision* as a function of *Recall* for different thresholds, where the former is defined as follows: $Precision = \frac{TP}{TP+FP}$.

Unlike the *AUC-ROC*, the *Precision-Recall* curve considers via the *Precision* the confidence in a positive prediction which can play a key role in imbalanced scenarios. The *Average Precision* (*AP*) is a summary of this confidence as the area under the curve [14, 83] and can be analytically computed as follows: $AP = \frac{1}{m^+} \sum_{i=1}^{m^+} p(k_i)$, where $p(k_i)$ is the *Precision* at the rank k_i corresponding to the i^{th} positive in the ranking. It is worth noting that *AP* can also be defined as

$$AP = \sum_t (r_t - r_{t-1}) p_t, \tag{4.2}$$

with r_t and p_t the *Recall* and *Precision*, respectively, at the t^{th} threshold of distinct prediction values. It has been shown in [19] that *AP* focuses more on the top of the ranking, contrary to the *AUC-ROC* which takes equally into account the entire ranking and tries to move up as many positives as possible. This phenomenon is illustrated in Figure 4.1 where two rankings composed of 4 positives (in blue) and 6 negatives (in grey) are compared in terms of *AUC-ROC* and *AP*. We can note that for both situations, *AUC-ROC* = 0.5, while *AP* is higher on the right than on the left case. Therefore *AUC-ROC* is not able to distinguish the two situations while *AP* highly prefers a scenario where two positives are ranked at the very top of the list even if this is at the expense of missing the two remaining positives. It turns out that this latter behavior can be very useful in situations where the constraints related to the application at hand require to focus on the first part of the ranking. This is the case when the budget in terms of number of allowed checks by human controllers is limited, like in bank or tax fraud detection.

The capacity of a system to optimize the number of positives at the very top of the ranking can be explicitly measured with a third criterion, called *Precision@k*. This measure corresponds to the number of positives among the top k of the ranking. As the choice of k for a given application can vary over time, so requiring to re-train the model, it might appear much more convenient to directly optimize the *AP* as a surrogate of this measure. This is what we suggest to do in our tree-based algorithm. We will see in the experiments that maximizing *AP* presents the nice property of optimizing at a cheaper cost the *Precision@k*.

4.3 Related Work

In this section, we present different methods that can be used for learning to rank from imbalanced datasets by inferring tree-based models, in the form of decision trees, meta-trees, random forests and tree ensembles. While the first two directly provide explicit decision rules, random forests and tree ensembles learned by gradient boosting are not readily interpretable, so reducing their benefit in applications like fraud detection despite the fact that they constitute the state of the art.

Decision Trees Although decision trees (DT), like CART [72], ID3 [90] or C4.5 [91], have been originally designed to address classification tasks, the discrete predictions can be used to establish a ranking. The splitting decision of CART is based on the minimization of the *Gini* impurity I_G. Assuming that the class label takes its value in the discrete set $\{1, 2, ..., k\}$,

Figure 4.1: Evaluation of the *AUC-ROC* and *AP* on two rankings. Blue (*resp.* grey) lines represent positive (*resp.* negative) samples. While *AUC-ROC* behaves similarly on both cases ($AUC\text{-}ROC = \frac{1}{2}$), the average precision is equal to 0.43 on the left and 0.68 on the right, illustrating that *AP* favors the ranking that put (at least some) positives at the very top of the list.

and that f_i denotes the fraction of the elements of the set with label i, I_G is defined as $I_G = \sum_{i=1}^{k} f_i(1 - f_i) = 1 - \sum_{i=1}^{k} f_i^2$ which is minimal when the leaf is pure, *i.e.* only composed of samples of the same class. On the other hand, ID3 and C4.5 make use of the information gain, based on the Shannon entropy. This latter allows to measure the disorder in a set and thus to select the threshold of the split that maximizes the information gain I_E defined as $I_E = -\sum_{i=1}^{k} f_i \log_2 f_i$.

Whatever the splitting criterion, once the decision tree is induced, several strategies can be applied to get a scoring function providing a ranking. First, the local probability at each leaf of being positive can be used as a continuous score. The ranking is obtained by sorting the leaves according to this score. On the other hand, in [5], the authors suggest to assign a score to each example of a leaf based on its distance to the boundaries of the hypercube representing the leaf and induced by the DT. The score is defined as the distance to the boundaries for every leaf with a majority of positive examples, or minus the distance in case of a majority of negative samples. Like in Support Vector Machines, this signed distance is exploited to rank the examples. Finally, in [73], the authors introduce a confusion factor at each internal node and use it to weight the contribution of the leaves and get the probability of an example.

Note that all the aforementioned methods share the same property. The optimization of the ranking is not part of the learning process. It is obtained by a post-process after the induction of the tree. In order to be efficient when addressing imbalanced settings, note also that the hyperparameters of the previous DT-based methods can be tuned by maximizing the *AUC-ROC*, *AP* or *Precision@k*, as introduced in Section 4.2.

The next paragraph introduces *TreeRank* [27], a learning to rank algorithm that has been specifically designed to generate a ranking that directly optimizes the *AUC-ROC*.

Meta-Trees *TreeRank* [27] is a decision tree algorithm that comes with theoretical guarantees for the bipartite ranking problem. A tree of trees (referred to as a meta-tree) is learned by directly optimizing the *AUC-ROC*. The principle is as follows. First, *TreeRank* induces a classical decision tree using a splitting procedure that takes into account the class imbalance. Then, the leaves are ordered according to a criterion based on the *AUC-ROC* curve. More precisely, for each leaf, the ratio β/α is computed with β (*resp.* α) the number of positives (*resp.* negatives) in the leaf divided by the number of positives (*resp.* negatives) in the root node. By assigning the label $+1$ to the considered leaf and -1 to all the others, we can notice that β corresponds to TPR and α to FPR as defined in Eq. 4.1. Ordering the leaves according to the ratio β/α is equivalent to sorting them according to the slope coefficient of the tangent

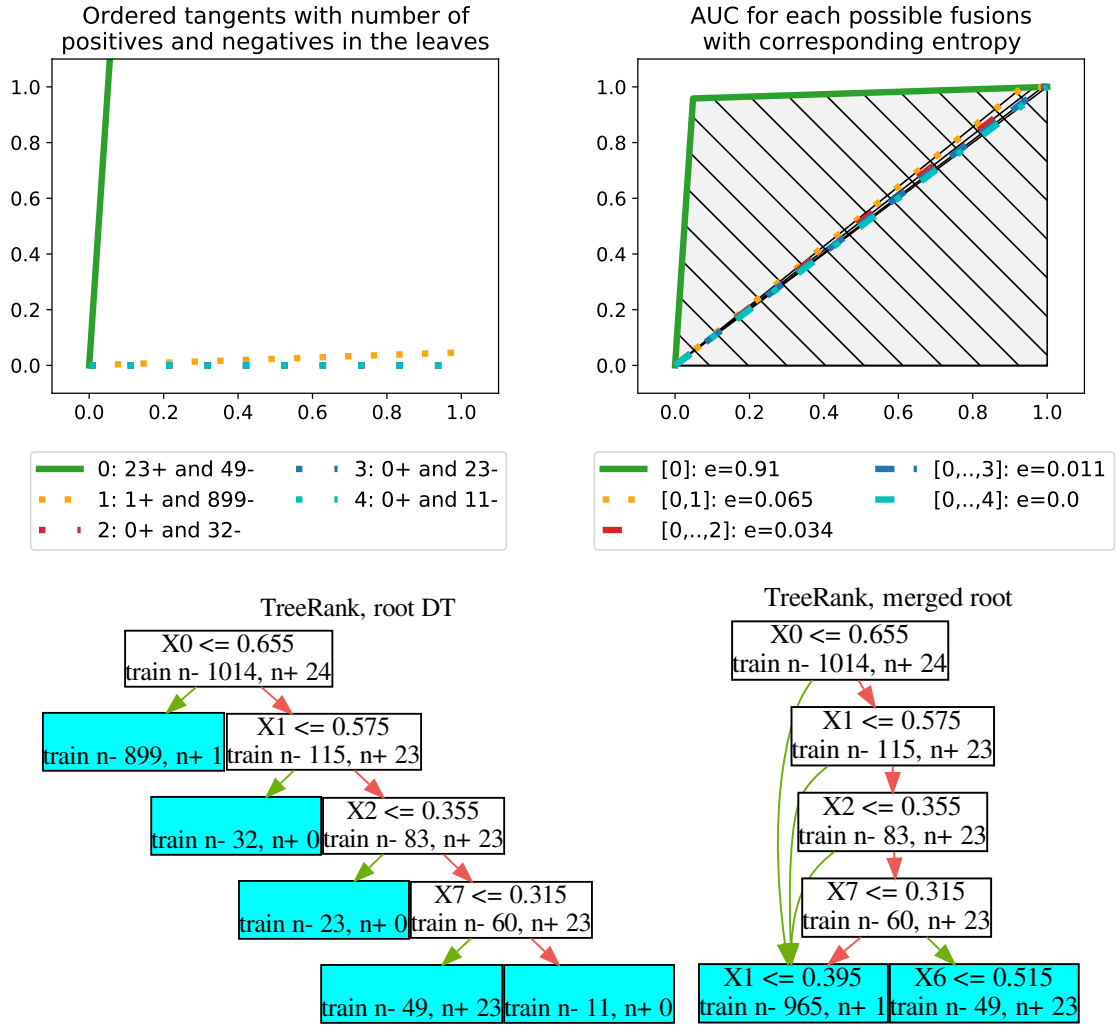

Figure 4.2: Illustration of **TreeRank** on the yeast6 dataset with maximum depth of 4: (top left) Some tangents to the ROC curve at 0 to order the leaves. (top right) Corresponding approximation of the ROC curve by a piece-wise function maximizing *AUC-ROC* by accumulating the sorted leaves from left to right.Here $max(entropy) = 0.91$ is achieved by putting the first leaf on the left and merging all the others on the right. (bottom left) Corresponding root's DT with splitting rules that maximize the *AUC-ROC*. (bottom right) Corresponding merged DT.

at the origin of the *ROC* curve, which boils down to maximizing the *AUC-ROC*. Then the leaves are partitioned into two sets, *left* and *right*, so that the partition maximizes an entropy defined as $\beta' - \alpha'$ where β' (*resp.* α') is the TPR (*resp.* FPR) of the left leaves. By doing so, *TreeRank* aims at finding the point with coordinates (α', β') maximizing the area under the piecewise function going from the origin $(0, 0)$ to the point (α', β') and then to the point $(1, 1)$. (see an illustration of the process in Figure 4.2).

The procedure is repeated by learning two new DTs, one for all leaves that fall on the left part and one for all the leaves that fall on the right. Finally, *TreeRank* assigns to each leaf a score based on the order in which the leaves were explored from left to right: the first leaf receives a score of 1 then the scores gradually decrease until the last leaf that receives a score of 1 divided by the total number of leaves.

Random Forests Random Forests (RF) [16, 59] aim to overcome the risk of overfitting of DTs by using a collection of partially independent trees. Each tree in the forest is still learned from n training examples, but the latter are randomly drawn (with replacement) from the training set S (bootstrap method). A sampling method is also performed over the features. At each node, the splits are optimized according to a subset of features randomly selected from the original feature space. A majority vote is finally applied from the predictions of these DTs. Note that the reduction of the variance in RFs is obtained at the price of loosing the interpretability of the induced model. In [26], the authors introduced a RF variant of TreeRank, called *Ranking Forest*, which aggregates Meta-trees and combines the corresponding rankings to optimize the *AUC-ROC*.

Gradient Tree Boosting Gradient Tree Boosting (GB) [48] is an ensemble method that consists in aggregating DTs, fitted sequentially on the residuals of the linear combination obtained at the previous step. In [43], the authors use a surrogate of the *Average Precision* (AP) that can be directly optimized in XGBoost [25], considered as one of the currently most effective GB methods. The resulting algorithm has been shown to be very efficient to address a highly imbalanced bank fraud detection task. Like Random Forest, the drawback of GB comes from its difficulty to induce a model that is directly interpretable.

4.4 MetaAP

In this section, we present our algorithm **MetaAP**. Inspired by *TreeRank*, its peculiarity comes from the optimization of the *Average Precision* (AP) instead of the *AUC-ROC*. The objective is to benefit from the capacity of AP to focus more on the top of the ranking to induce an interpretable model useful in applications where the budget in terms of human controllers is limited, like in fraud detection.

Let n be the number of examples of a given node of the current tree and n^+ be the corresponding number of positive samples. Let us consider a splitting decision that would send n_l examples in the left child node and *resp.* n_r in the right child node. We aim at selecting the split that would maximize $n_l AP_{left} + n_r AP_{right}$ where AP_{left} and AP_{right} are two approximations of the *Average Precision*. AP_{left} is computed as if all the examples in the left child are classified $+1$ and all the examples in the right child are classified -1. Conversely, AP_{right} is calculated as if all the examples in the right child are classified $+1$ and all the examples in the left node are classifies -1. We further define n_l^+ and n_r^+ as the number of positive examples in the left and right children. In this binary setting, three thresholds are used in the computation of AP_{left} with the associated *Recall* and *Precision* defined as follows:

threshold	t_0	t_1	t_2
Recall	0	$\frac{n_l^+}{n^+}$	1
Precision	1	$\frac{n_l^+}{n_l}$	$\frac{n^+}{n}$

Plugging these values in Equation (4.2), we get:

$$AP_{left} = \frac{n_l^+}{n^+}\frac{n_l^+}{n_l} + \left(1 - \frac{n_l^+}{n^+}\right)\frac{n^+}{n} = \frac{n_l^{+2}}{n^+ n_l} + \frac{n_r^+}{n}$$

and similarly $AP_{right} = \dfrac{n_r^{+2}}{n^+ n_r} + \dfrac{n_l^+}{n}$.

In addition to being easy to compute, this splitting criterion allows us to better infer pure positive leaves that will lead to a better global AP (see Figure 4.3, bottom left).

Once a decision tree is learned (called DTAP as shown in Figure 4.3, bottom left), we need to order the leaves as done in *TreeRank*. However, since we aim at maximizing AP, we have to work with the *Precision-Recall* (PR) curve instead of the *ROC* curve (see Section 4.2), which is more challenging for two main reasons. First, the PR curve is not increasing on $[0; 1]$ from 0 to 1. It decreases from 1 to a value corresponding to the positive rate of the dataset. To keep the same strategy as *TreeRank*, we rather minimize the area under the (1-P)R curve which boils down to maximizing the area under the PR curve (Figure 4.3, top right). Second, unlike the *ROC* curve, the (1-P)R is not a monotonically increasing function, having local maxima each time negatives are highly ranked. To overcome this issue, our ordering strategy will relegate the leaves leading to such a scenario to the end of the list preventing them from being merged on the left part of the meta-tree, the part that really matter to get a high AP. More precisely, we rank in ascending order the directing coefficients of the tangents to the (1-P)R curve at 0 by calculating for each leaf the $(1 - Precision)/Recall$ ratio (see Figure 4.3, top left). This implies that leaves with a majority of positives will appear at the beginning of the list, those with the most positives first because leading to the lowest directing coefficient corresponding to a $Precision$ close to 1. Finally, according to the obtained ranking, the merging of the leaves into two subsets (the left and right parts of the meta-tree) is performed as follows: we accumulate the positives and negatives according to the ranking of the leaves and we select the merging threshold that maximizes the AP on the left (Figure 4.3, bottom right).

Note that similarly to DTs, it is possible to adapt our tree-based method to build random forests by training several times **MetaAP** from a certain number of bootstraps of the training set. We denote this adaptation **MetaAPForest** in the next section.

4.5 Experiments

In this section, we present the experiments conducted on 28 public datasets, more or less imbalanced, coming from the UCI[1] and KEEL[2] repositories. We also performed experiments on private datasets provided by the French Ministry of Economy and Finance (DGFiP).

4.5.1 Datasets and experimental setup

The main characteristics of the 28 public datasets are summarized in Table 4.1. The private datasets correspond to the tax and VAT declarations of French companies. They are used for the detection of (i) over-evaluated charges (called 1st fraud in the following) and (ii) international VAT frauds (2nd fraud). There are 40 datasets with an average of about 7,000 samples and 250 features. The imbalance ratio ranges from 0.3% to 24.3%. The main characteristics of these private datasets are summarized in Table 4.2 at the end of this Chapter. Note that each year, only 50,000 controls can be carried out from a panel of more than 3 million companies. Therefore, the output ranking must contain as many positives as possible in the very top list in order to optimize the recovery of sums due. Moreover, the model must be interpretable to justify why a company has been selected. The Python code of the algorithms, experiments, plots are publicly available[3].

For each dataset, we split the data into training (70%) and test (30%) sets. We select the hyperparameters by maximizing the AP through a 5-folds cross validation over the training set. We repeat the process over 20 runs and average the results in both terms of AP and $Precision@k$. For the latter, we select k as the percentage of the number of positives in the test set.

[1] `https://archive.ics.uci.edu/ml/datasets.html`
[2] `https://sci2s.ugr.es/keel/datasets.php`
[3] `https://github.com/LeoGautheron/submission-PRL`

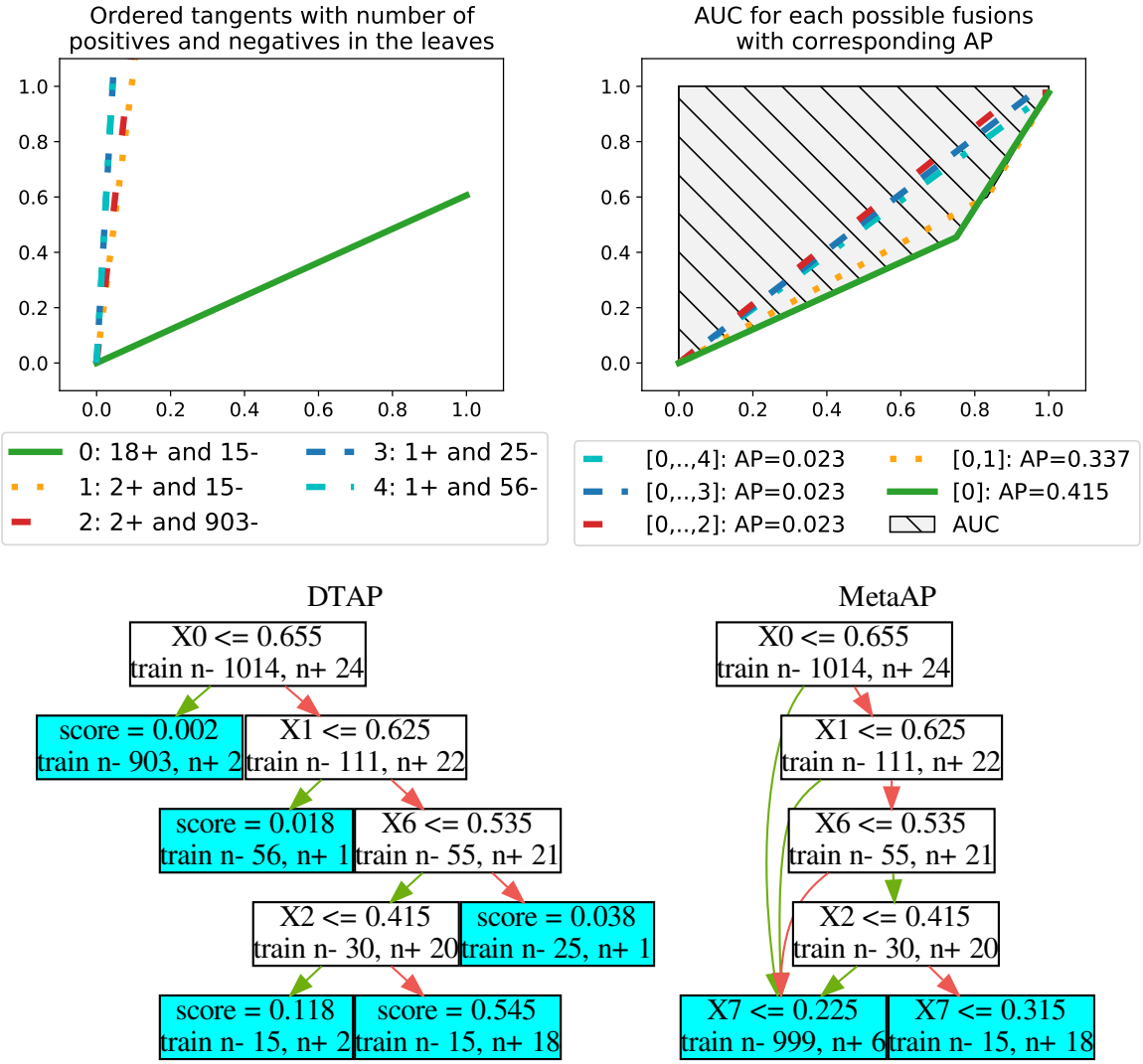

Figure 4.3: **MetaAP** run on *yeast*6 (one of the most imbalanced public dataset) with maximum depth of 4: (top left) Some tangents to the (1-P)R curve at 0 whose slopes are used to order the leaves. (top right) Corresponding approximation of the (1-P)R curve by a piece-wise function maximizing AP_{left} by accumulating the sorted leaves from left to right. MetaAP aims at maximizing the hatched area above the curve. Here $max(AP_{left}) = 0.415$ is achieved by putting the first leaf on the left and merging all the others on the right. (bottom left) Corresponding root's DT with splitting rules that maximize the AP. (bottom right) Corresponding merged DT.

4.5.2 Comparison with Decision Tree methods

We carried out a first series of experiments consisting in comparing **MetaAP** to three DT baselines: the ranking DT algorithm [5] using either (i) the **Gini** ot (ii) the **Entropy** criterion, and (iii) **TreeRank** [27]. The depths of internal trees and the global tree of **TreeRank** and **MetaAP** are tuned in the set $\{2, 3, ..., 9, 10\}$ which can lead when building the full tree to a maximum depth of $10 \times 10 = 100$. To make the experiments fair, the tree depths of **Gini** and **Entropy**-based DTs are tuned in the set $\{2, 3, ..., 8, 9, 10, 20, ..., 80, 90, 100\}$.

The mean APs over the 28 public datasets are reported in Figure 4.4. It shows five comparisons from 50% to 10%: the former encompasses the datasets with at most an imbalance ratio of 50% (thus, considers all the 28 datasets), while the latter takes into account the 6 most

Table 4.1: Information about the public datasets, sorted by positive rate. %+: percentage of positives, n: number of examples, d: number of features.

datasets	% +	n	d	datasets	% +	n	d
abalone20	00.62%	4177	10	newthyroid	30.23%	215	5
abalone17	01.39%	4177	10	glass	32.71%	214	9
yeast6	02.36%	1484	8	wine	33.15%	178	13
redwinequality4	03.31%	1599	11	pima	34.90%	768	8
libras	06.67%	360	90	ionosphere	35.90%	351	34
satimage	09.73%	6435	36	wdbc	37.26%	569	30
pageblocks	10.23%	5473	10	autompg	37.50%	392	7
yeast3	10.98%	1484	8	spambase	39.42%	4597	57
bankmarketing	11.70%	45211	51	bupa	42.03%	345	6
abalone8	13.60%	4177	10	heart	44.44%	270	13
segmentation	14.29%	2310	19	australian	44.49%	690	14
hayes	22.73%	132	4	balance	46.08%	625	4
vehicle	23.52%	846	18	sonar	46.63%	208	60
german	30.00%	1000	24	splice	48.09%	3175	60

imbalanced datasets with at most 10% of positives. Whatever the percentage between 50% and 10%, we can note that our method (in green) outperforms the other competitors. The superiority of **MetaAP** seems to be even larger as the imbalance ratio increases. In order to evaluate the significance of these results, we performed a Wilcoxon signed-rank test in each scenario by comparing **MetaAP with the first best competitor**. It is worth noting that at 50%, thus comparing with **TreeRank** over the 28 datasets, we obtain a p-value smaller than 0.05. For the other imbalance ratios, we get a p-value between 0.1 (for 10%) and 0.2 (for 20%, 30% and 40%).

The usefulness of our method is emphasized by Figure 4.7 (top) which reports the results in terms of $Precision@k$ (only for the cases 10%, 30% and 50% for the sake of conciseness). While, as expected, the gap between the methods in terms $Precision@k$ tends to be smaller and smaller as the percentage of positives grows, for small values of k (the ones that are considered in the case of budget limitation), **MetaAP** is most of the time much better that **TreeRank** and all the other competitors (green curve above the others). This behavior is confirmed and even amplified on the fraud detection task. The results in terms of $Precision@k$ reported in Figure 4.5 show that **MetaAP** significantly outperforms all the other methods for the two cases of studied frauds up to a value of k equivalent to 25% of the number of positives in the test set. This impressive result opens the door to the use of **MetaAP** for making the tax audit process much more efficient.

4.5.3 Analysis of an early stopping strategy

To analyze the impact of the tree depth on the behavior of the methods, we studied an early stopping in the construction of the models at fixed depths p from 1 to 10 for **MetaAP** and **TreeRank**. For the sake of fairness, we used a depth of p^2 for **Gini** and **Entropy**-based DTs which leads to the same expressiveness for all the resulting (meta)-trees. The results on the public datasets are reported in Figure 4.6. We can note that as soon as **MetaAP** has a depth greater than 3, it outperforms all the other methods, at equivalent depth, whatever the percentage of positives considered.

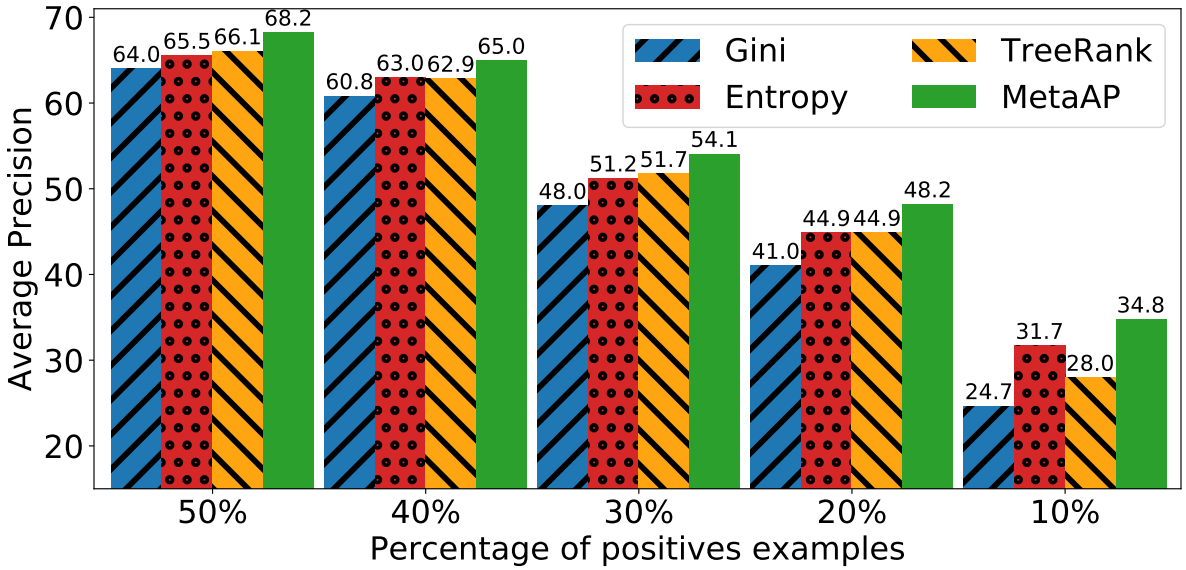

Figure 4.4: Mean *AP* as a function of the percentage of positives.For 50%, *AP* is computed from all the 28 datasets. For 10%, only 6 datasets are used.

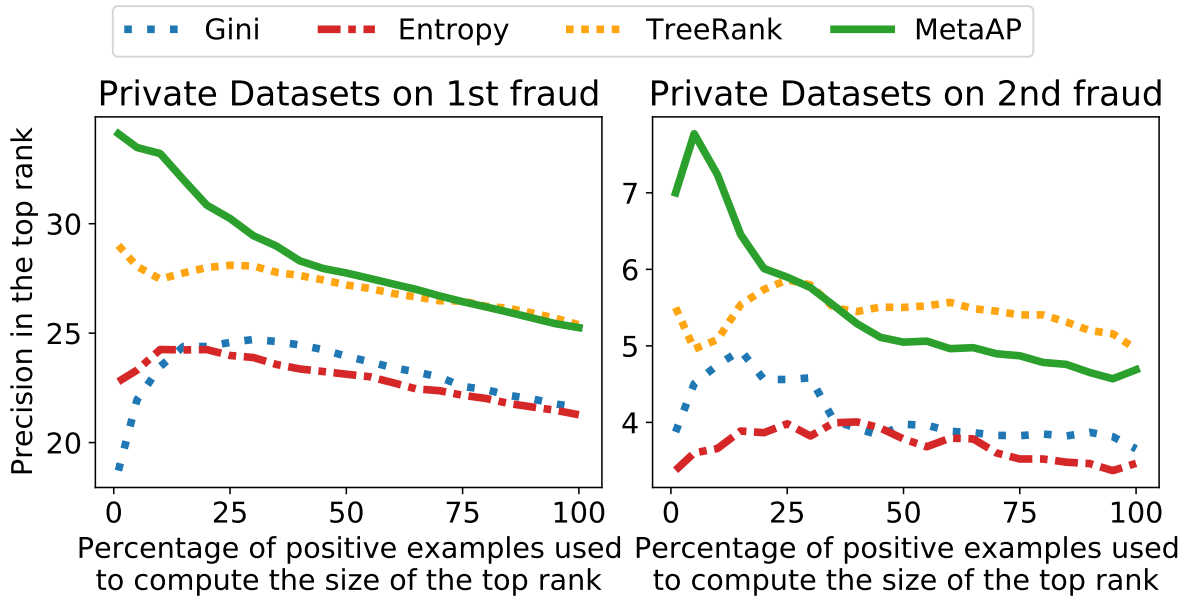

Figure 4.5: *Precision@k* according to the % of positives in the private datasets.

4.5.4 Comparison with forest-based methods

In a last series of experiments, even though we are mainly interested in interpretable models, we compared our Random Forest version **MetaAPForest** with 3 baselines: (i) the RF **Entropy-Forest** using the **Entropy** DT [5], (ii) the ranking version of *XGBoost* [25]: **XGBRanker**, (iii) **SGBAP** [43] that optimizes the *AP*. The number of DTs in the forests and in the gradient tree boosting methods was set to 100 and the depth of the trees is tuned between 2 and 10. For the forests, we considered a bootstrap of the size of the number of training examples for each tree, but we did not use bootstrap for the features and considered all of them at each node. We did not make use of the RF version of *TreeRank* because its implementation in R has been shown to be too much memory consuming during the experiments. We neither considered **GiniForest**

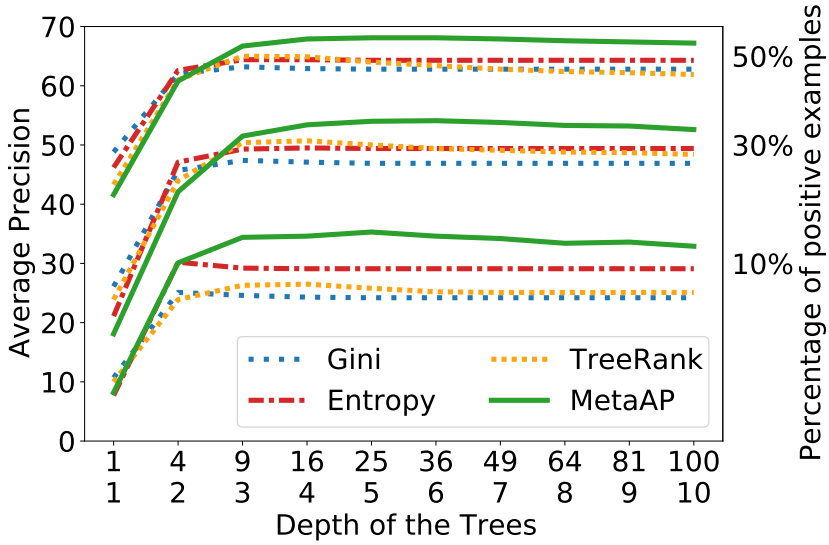

Figure 4.6: Mean AP as a function of the depth on the public datasets with a proportion of positives of at most 10%, 30% and 50%. The 1st line (*resp.* 2nd) of the x-axis represents the depth of the **Gini** and **Entropy**-based DTs (*resp.* **TreeRank** and **MetaAP**).

because it led to slightly worse performances than its counterpart **EntropyForest**.

As expected, Gradient Boosting outperforms the other approaches in terms of AP. Over the 28 public datasets, **XGBRanker** reaches 77.0% while **MetaAP** leads to an AP equal to 74.8%, 74.9% for **EntropyForest** and 73.3% for **SGBAP**. However, recall that both Gradient Boosting and Random Forest methods do not fulfill our requirement of inferring interpretable models. Interestingly, Figure 4.7 (bottom) shows that in terms of $Precision@k$, **MetaAP** is competitive by outperforming both **EntropyForest** and **SGBAP** and being just slightly below **XGBRanker**.

4.5.5 Compact and interpretable meta-trees

We end this section by illustrating in Figure 4.8 the fact that **MetaAP** allows to induce much more compact models with the same fixed depth than standard DT methods. This compression comes from the fact that by construction **MetaAP** allows leaves to be reached by several paths. In terms of AP, we can see that such compact meta-trees can allow to avoid overfitting phenomena as illustrated in the Figure by a smaller gap between the APs at training and test time (49.82 versus 43.68) compared to that of the Entropy-based DT (58.62 versus 35.16).

4.6 Conclusion and Perspectives

In this Chapter, we address the challenging problem of learning to rank from imbalanced data. We design an algorithm that builds meta-trees by optimizing during the learning process the Average Precision (AP). As far as we know, this is the fist attempt to optimize directly this non-convex and non separable function in a tree-based ranking method. The resulting algorithm **MetaAP** shows very promising results on public datasets and on a tax fraud detection task where the need of generating a short list of alerts maximizing the AP is of great interest. **MetaAP** also comes with the valuable property of inducing compact interpretable models. In future work, we plan to investigate the optimization of the $Precision@k$. This would allow us to directly take into account the number of controls k allowed by the application at hand. However, we will need to figure out solutions, for example using transfer learning methods, to

avoid retraining from scratch a model learned for a given k. Finally, even if our setting in not as favourable as that of **TreeRank** (unlike the ROC curve, the (1-P)R is not a monotonically increasing function), we will investigate the possibility to derive generalization guarantees.

Table 4.2: Information about the private datasets with %+ the percentage of positives, n the number of examples and d the number of features.

datasets	% +	n	d	datasets	% +	n	d	datasets	% +	n	d
dgfip1_1	11.1	4794	249	dgfip8_1	17.8	1028	255	dgfip15_1	21.6	7370	273
dgfip1_2	00.4	4317	241	dgfip8_2	02.2	872	244	dgfip15_2	01.9	5960	265
dgfip2_1	17.0	10033	264	dgfip9_1	16.4	409	171	dgfip16_1	10.3	789	162
dgfip2_2	00.9	8460	257	dgfip9_2	01.2	346	154	dgfip16_2	01.0	723	154
dgfip3_1	14.1	2026	230	dgfip10_1	13.0	7102	246	dgfip17_1	17.2	15658	302
dgfip3_2	00.3	1756	226	dgfip10_2	00.6	6252	244	dgfip17_2	00.9	13186	300
dgfip4_1	16.3	240	76	dgfip11_1	20.0	14209	328	dgfip18_1	11.5	2225	260
dgfip4_2	01.9	208	65	dgfip11_2	02.9	11964	326	dgfip18_2	01.9	2023	256
dgfip5_1	16.4	21717	321	dgfip12_1	21.4	4244	252	dgfip19_1	24.3	14377	262
dgfip5_2	03.8	19117	318	dgfip12_2	02.8	3477	251	dgfip19_2	01.3	11124	255
dgfip6_1	14.5	5006	229	dgfip13_1	13.5	11114	268	dgfip20_1	16.5	19867	294
dgfip6_2	04.6	4499	228	dgfip13_2	00.4	9710	262	dgfip20_2	05.0	17584	294
dgfip7_1	12.2	2314	229	dgfip14_1	16.2	4344	262				
dgfip7_2	01.4	2068	225	dgfip14_2	01.4	3718	254				

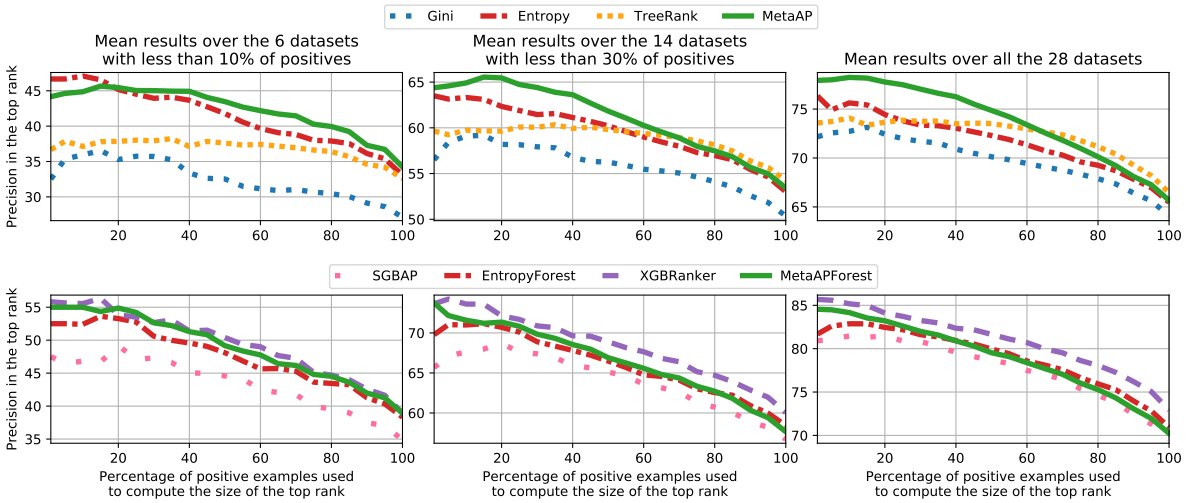

Figure 4.7: *Precision@k* according to the percentage of positives.(Top) Results for DT methods. (Bottom) Results for RF and Gradient Tree Boosting methods.

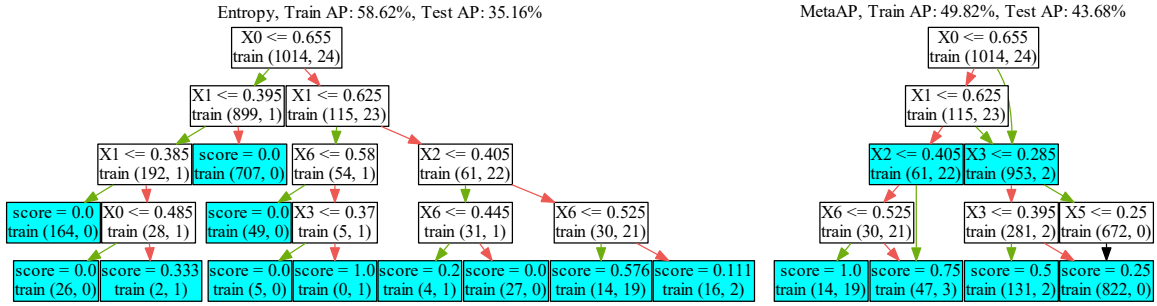

Figure 4.8: Trees learned on *yeast6* using the **Entropy** (left) and **MetaAP** (right). A green (*resp.* red) arrow means that the feature considered in the node is lower or equal to (*resp.* larger than) the threshold. A black arrow is used when both the green and red arrows are going from the same parent to the same child. Each node (*resp.* leave) contains the splitting criterion (*resp.* the ranking score) and the number of training negatives and positives (n^-, n^+).

Conclusion and Perspectives

In this manuscript, we presented new supervised methods for addressing the challenging task of highly imbalanced learning. This thesis has been carried out in the context of a partnership with the French Ministry of Economy and Finance. Our contributions aimed at tackling the complex task of fraud detection characterized by the fact that fraudsters often try to have a behavior as close to non frausters as possible in order to remain undetected. Thus, unlike anomalies, suspicious positive examples are often hidden in the middle of many negative samples. That is why detecting frauds can be seen as looking for a needle in a haystack. A second peculiarity of fraud detection comes from the limitation of the budget in terms of human controllers. This constraint prompts us to design specific models that minimize the number of false negatives and reduce as much as possible the number of alerts sent to the agents. The last requirement we had to take into account in this thesis is the need for the DGFiP to get models that are interpretable so as to support human decision making.

Based on the assumption that fraudsters try to mimic non-fraudsters and are much scarcer than the latter, we have developed two strategies that force the model to pay more attention to the positive examples.

With $\boldsymbol{\gamma k}-\mathbf{NN}$, we adapted the Nearest-Neighbor algorithm so that it favors positive examples. By weighting the distances to the latter with a coefficient (smaller than 1), we artificially bring the examples of interest closer to the query, thus simulating an expansion of the Voronoi cells surrounding them. Through an extensive experimental study, this simple and natural method has been shown to be very efficient. It is worth noting that this first contribution has been deployed at the DGFiP and is currently used to improve VAT fraud detection algorithms. A promising perspective related to this first contribution would be to extend the idea of the local variant of $\boldsymbol{\gamma k}-\mathbf{NN}$ (see Section 2.5.4) to the design of a multi-view learning approach, where the different results of $\boldsymbol{\gamma k}-\mathbf{NN}$ obtained with complementary subsets of features (the different views) would be combined in some way. Another interesting perspective would consist in working on a probabilistic version of $\boldsymbol{\gamma k}-\mathbf{NN}$ allowing us to prioritize the examples to be taken into consideration. The exploitation of γ in the definition of such probabilities has not been studied yet and could be relevant. Finally, as $\boldsymbol{\gamma k}-\mathbf{NN}$ artificially extends the Voronoi cells around the positive examples, it mimics in some way the behavior of a generative model that would create artificial examples in some particular part of the feature space. Establishing the link between $\boldsymbol{\gamma k}-\mathbf{NN}$ and GANs would be a promising line of research.

With **MLFP**, a metric learning algorihm, we extended our geometrical analysis by learning ellipsoids that allow us to better catch positive examples. The idea was to correct the main drawback of standard Metric Learning methods which focus too much on the majority class. We addressed this issue by optimizing a Mahalanobis distance under a constraint on the largest eigenvalue of the PSD matrix that allows us to control False Negative rate. Exploiting the uniform stability framework, **MLFP** comes with generalization guarantees. One limitation of **MLFP** comes from the assumption that a linear transformation is sufficient to enhance the feature representation. To overcome this problem, different strategies deserve to be studied. The first one would consist in kernelizing **MLFP**, which is not straightforward because requiring

to rewrite the Mahalanobis distance-based problem into inner products. One could also design a deep learning approach at the expense of the generalization guarantees. A more promising strategy would consist in learning different local metrics for different regions of the input space. Finally, combining a Mahalanobis distance with a sampling strategy might lead to a new family of imbalanced learning methods.

In response to the request of the DGFiP, the third contribution of this thesis was guided by the need of interpretable models to support human decision making. We presented **MetaAP**, a ranking algorithm that builds Meta-Decision Trees by directly optimizing the *Average Precision* at each step of the process. **MetaAP** aims at pushing positive examples at the very top of the ranking allowing the limited number of agents to control the alerts. A natural extension of this work is the optimization of the $precision@k$ which seems to be more interesting in the context of fraud detection when one has a known constrained budget. However, in such a scenario, how to take into account the parameter "$@k$" in the construction of the meta-tree? How to avoid retraining from scratch the model learned each time the number of agents changes? These are open questions.

Other perspectives are directly related to the activities carried out at the DGFiP where I am now working full time. For instance, I am expected to work on other settings, like the detection of frauds on real estate transfers. Currently, the model in place at the DGFiP uses a Nearest-Neighbor algorithm to predict the market value of real estate, in order to know if the sales are suspicious. The idea would be to push the research a step further and detect fraudulent sales by using a more adapted representation space, for which the formalism remains to be defined. Another research axis would be the detection of fraud in graphs. Indeed, the DGFiP has recently started to link the different information in its possession. Given the fraudsters already detected, it could be interesting to look at the interactions between them and the possible propagation of frauds in this graph. On the other hand, considering the multiplicity of data sources at my disposal, I will have to work on dimensionality reduction and feature engineering with the requirement to keep the models interpretable. Finally, I plan to work on semi-supervised learning techniques. Indeed, considering the difficulty to obtain a significant number of labeled examples and the selection bias inherent to the process, it seems appropriate to benefit from the huge amount of unlabeled data at my disposal to improve the whole chain of controls.

Bibliography

[1] An experiment with the edited nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(6):448–452, 1976. 34

[2] Aisha Abdallah, Mohd Aizaini Maarof, and Anazida Zainal. Fraud detection system: A survey. *Journal of Network and Computer Applications*, 68:90–113, 2016. 1, 67, 89

[3] Charu C. Aggarwal. *Outlier Analysis*. Springer International Publishing, 2017. 46, 67

[4] Shikha Agrawal and Jitendra Agrawal. Survey on anomaly detection using data mining techniques. In Liya Ding, Charles Pang, Mun-Kew Leong, Lakhmi C. Jain, and Robert J. Howlett, editors, *KES*, volume 60 of *Procedia Computer Science*, pages 708–713. Elsevier, 2015. 90

[5] Isabelle Alvarez and Stephan Bernard. Ranking cases with decision trees: a geometric method that preserves intelligibility. In *IJCAI*, pages 635–640. Citeseer, 2005. 93, 97, 99

[6] Ricardo Barandela, José Salvador Sánchez, V Garca, and Edgar Rangel. Strategies for learning in class imbalance problems. *Pattern Recognition*, 36, 2003. 49, 55

[7] Kevin Bascol, Rémi Emonet, Élisa Fromont, Amaury Habrard, Guillaume Metzler, and Marc Sebban. From cost-sensitive to tight f-measure bounds. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019*, volume 89, pages 1245–1253. PMLR, 2019. 46

[8] Richard A Bauder, Taghi M Khoshgoftaar, and Tawfiq Hasanin. Data sampling approaches with severely imbalanced big data for medicare fraud detection. In *2018 IEEE 30th international conference on tools with artificial intelligence (ICTAI)*, pages 137–142. IEEE, 2018. 46

[9] Aurélien Bellet, Amaury Habrard, and Marc Sebban. A survey on metric learning for feature vectors and structured data. *CoRR*, abs/1306.6709, 2013. 47, 49, 68, 69

[10] Aurélien Bellet, Amaury Habrard, and Marc Sebban. *Metric Learning*, volume 2015 of *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool Publishers, 2015. 16, 27, 47, 49, 68, 72, 81

[11] Joseph Berkson. Application of the logistic function to bio-assay. *Journal of the American Statistical Association*, 39(227):357–365, 1944. 7, 18

[12] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992. 7, 16, 17

[13] Olivier Bousquet and André Elisseeff. Stability and generalization. *Journal of machine learning research*, 2(Mar):499–526, 2002. 11, 12, 70, 72, 81

[14] Kendrick Boyd, Kevin H Eng, and C David Page. Area under the precision-recall curve: point estimates and confidence intervals. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 451–466. Springer, 2013. 92

[15] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996. 23

[16] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. 23, 95

[17] K. H. Brodersen, C. S. Ong, K. E. Stephan, and J. M. Buhmann. The balanced accuracy and its posterior distribution. In *20th International Conference on Pattern Recognition*. IEEE, 2010. 46, 48

[18] Sébastien Bubeck and Nicolo Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *arXiv preprint arXiv:1204.5721*, 2012. 7

[19] Christopher JC Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581):81, 2010. 90, 92

[20] Ramiro Camino and Chris Hammerschmidt. Oversampling tabular data with deep generative models: Is it worth the effort? *"I Can't Believe It's Not Better!"NeurIPS 2020 workshop*, 2020. 90

[21] Lidia Ceriani and Paolo Verme. The origins of the gini index: extracts from variabilità e mutabilità (1912) by corrado gini. *The Journal of Economic Inequality*, 10(3):421–443, 2012. 21

[22] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 2009. 46

[23] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16, 2002. 37, 46, 90

[24] Nitesh V Chawla, Aleksandar Lazarevic, Lawrence O Hall, and Kevin W Bowyer. Smoteboost: Improving prediction of the minority class in boosting. In *European conference on principles of data mining and knowledge discovery*, pages 107–119. Springer, 2003. 42

[25] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, et al. Xgboost: extreme gradient boosting. *R package version 0.4-2*, 1(4), 2015. 25, 90, 95, 99

[26] Stéphan Clémençon, Marine Depecker, and Nicolas Vayatis. Ranking forests. *Journal of Machine Learning Research*, 14(Jan):39–73, 2013. 95

[27] Stéphan Clémençon and Nicolas Vayatis. Tree-based ranking methods. *IEEE Transactions on Information Theory*, 55(9):4316–4336, 2009. 90, 93, 97

[28] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995. 7, 16

[29] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967. 14, 46

[30] Jason V. Davis, Brian Kulis, Prateek Jain, Suvrit Sra, and Inderjit S. Dhillon. Information-theoretic metric learning. In *Proceedings of the 24th international conference on Machine learning*, pages 209–216. ACM, 2007. 29, 47, 49, 68, 69, 76

[31] Cour des Comptes. La fraude aux prélèvements obligatoires. 2019. 1

[32] Shuya Ding, Bilal Mirza, Zhiping Lin, Jiuwen Cao, Xiaoping Lai, Tam V. Nguyen, and Jose Sepulveda. Kernel based online learning for imbalance multiclass classification. *Neurocomputing*, 277:139–148, 2018. 68

[33] Georgios Douzas and Fernando Bação. Effective data generation for imbalanced learning using conditional generative adversarial networks. *Expert Syst. Appl.*, 91:464–471, 2018. 90

[34] Sahibsingh A Dudani. The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, 4, 1976. 14, 48, 55

[35] Sri Harsha Dumpala, Rupayan Chakraborty, and Sunil Kumar Kopparapu. A novel data representation for effective learning in class imbalanced scenarios. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 2100–2106, 7 2018. 68

[36] Charles Elkan. The foundations of cost-sensitive learning. In *International joint conference on artificial intelligence*, pages 973–978. Morgan Kaufmann, 2001. 40, 52, 90

[37] RP Espíndola and NFF Ebecken. On extending f-measure and g-mean metrics to multi-class problems. *WIT Transactions on Information and Communication Technologies*, 35, 2005. 48

[38] Wei Fan, Salvatore J Stolfo, Junxin Zhang, and Philip K Chan. Adacost: misclassification cost-sensitive boosting. In *Icml*, volume 99, pages 97–105. Citeseer, 1999. 41

[39] Lin Feng, Huibing Wang, Bo Jin, Haohao Li, Mingliang Xue, and Le Wang. Learning a distance metric by balancing kl-divergence for imbalanced datasets. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, PP:1–12, 01 2018. 68, 70, 90

[40] Alberto Fernández, Salvador Garcia, Francisco Herrera, and Nitesh V Chawla. Smote for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary. *Journal of Artificial Intelligence Research*, 61, 2018. 46

[41] César Ferri, José Hernández-Orallo, and R Modroiu. An experimental comparison of performance measures for classification. *Pattern Recognition Letters*, 30, 2009. 46

[42] Evelyn Fix and Joseph Lawson Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique*, 57(3):238–247, 1989. 7

[43] Jordan Fréry, Amaury Habrard, Marc Sebban, Olivier Caelen, and Liyun He-Guelton. Efficient top rank optimization with gradient boosting for supervised anomaly detection. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18-22, 2017, Proceedings, Part I*. Springer, 2017. 46, 90, 95, 99

[44] Jordan Frery, Amaury Habrard, Marc Sebban, and Liyun He-Guelton. Non-linear gradient boosting for class-imbalance learning. In *2nd International Workshop on Learning with Imbalanced Domains: Theory and Applications*, pages 38–51, 2018. 68

[45] Yoav Freund. Boosting a weak learning algorithm by majority. *Information and computation*, 121(2):256–285, 1995. 7

[46] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997. 23, 25

[47] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001. 23, 25

[48] Jerome H Friedman. Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4):367–378, 2002. 23, 25, 95

[49] Mikel Galar, Alberto Fernandez, Edurne Barrenechea, Humberto Bustince, and Francisco Herrera. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(4):463–484, 2011. 41, 90

[50] Léo Gautheron, Amaury Habrard, Emilie Morvant, and Marc Sebban. Metric learning from imbalanced data with generalization guarantees. *Pattern Recognit. Lett.*, 133:298–304, 2020. 90

[51] Léo Gautheron, Emilie Morvant, Amaury Habrard, and Marc Sebban. Metric learning from imbalanced data. In *arXiv*. 1909.01651, 2019. 68, 69, 76

[52] Sunder Gee. *Fraud and fraud detection: a data analytics approach*. 2014. 48

[53] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. 2014. 46

[54] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. Borderline-smote: a new over-sampling method in imbalanced data sets learning. In *International conference on intelligent computing*. Springer, 2005. 37

[55] Peter Hart. The condensed nearest neighbor rule (corresp.). *IEEE transactions on information theory*, 14(3):515–516, 1968. 34

[56] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. IEEE, 2008. 37

[57] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009. 90

[58] Shohei Hido, Hisashi Kashima, and Yutaka Takahashi. Roughly balanced bagging for imbalanced data. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 2(5-6):412–426, 2009. 42

[59] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995. 23, 95

[60] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933. 7

[61] Chen Huang, Yining Li, Chen Change Loy, and Xiaoou Tang. Learning deep representation for imbalanced classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5375–5384, 2016. 68

[62] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017. 25

[63] Salman H. Khan, Munawar Hayat, Mohammed Bennamoun, Ferdous A. Sohel, and Roberto Togneri. Cost-sensitive learning of deep feature representations from imbalanced data. *IEEE transactions on neural networks and learning systems*, 29(8):3573–3587, 2017. 68

[64] Aryeh Kontorovich, Sivan Sabato, and Ruth Urner. Active nearest-neighbor learning in metric spaces. In *Advances in Neural Information Processing Systems 29*. NIPS, 2016. 46

[65] Aryeh Kontorovich and Roi Weiss. A Bayes consistent 1-NN classifier. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38. PMLR, 2015. 46

[66] Miroslav Kubat and Stan Matwin. Addressing the curse of imbalanced training sets: One-sided selection. In Douglas H. Fisher, editor, *Proceedings of the Fourteenth International Conference on Machine Learning (ICML)*, pages 179–186. Morgan Kaufmann, 1997. 34, 48

[67] Brian Kulis et al. Metric learning: A survey. *Foundations and Trends® in Machine Learning*, 5(4):287–364, 2013. 68, 69

[68] Jorma Laurikkala. Improving identification of difficult small classes by balancing class distribution. In Silvana Quaglini, Pedro Barahona, and Steen Andreassen, editors, *Artificial Intelligence in Medicine*, pages 63–66, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. 34

[69] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. 54

[70] Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999. 7

[71] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017. 38, 56

[72] Breiman LI, Jerome Friedman, RA Olshen, and C.J. Stone. *Classification and Regression Trees (CART)*, volume 40. 09 1984. 7, 20, 21, 92

[73] Charles X Ling and Robert J Yan. Decision tree with better ranking. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 480–487, 2003. 93

[74] Kun Liu, Jiangrui Han, Haiyong Chen, Haowei Yan, and Peng Yang. Defect detection on el images based on deep feature optimized by metric learning for imbalanced data. In *2019 25th International Conference on Automation and Computing (ICAC)*, pages 1–5. IEEE, 2019. 90

[75] Wei Liu and Sanjay Chawla. Class confidence weighted knn algorithms for imbalanced data sets. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 345–356. Springer, 2011. 48

[76] Xu-Ying Liu, Jianxin Wu, and Zhi-Hua Zhou. Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(2):539–550, 2008. 42

[77] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982. 7

[78] Ulrike von Luxburg and Olivier Bousquet. Distance-based classification with lipschitz functions. *Journal of Machine Learning Research*, 5, 2004. 46

[79] Inderjeet Mani and I Zhang. knn approach to unbalanced data distributions: a case study involving information extraction. In *Proceedings of workshop on learning from imbalanced datasets*, volume 126. ICML United States, 2003. 34

[80] Llew Mason, Jonathan Baxter, Peter Bartlett, and Marcus Frean. Boosting algorithms as gradient descent in function space. In *Proc. NIPS*, volume 12, pages 512–518, 1999. 23, 25

[81] Josey Mathew, Ming Luo, Chee Khiang Pang, and Hian Leng Chan. Kernel-based smote for svm classification of imbalanced datasets. In *IECON 2015-41st Annual Conference of the IEEE Industrial Electronics Society*, pages 001127–001132. IEEE, 2015. 68

[82] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943. 7

[83] Aditya Krishna Menon and Robert C Williamson. Bipartite ranking: a risk-theoretic perspective. *The Journal of Machine Learning Research*, 17(1):6766–6867, 2016. 92

[84] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015. 7

[85] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018. 7

[86] Hien M Nguyen, Eric W Cooper, and Katsuari Kamei. Borderline over-sampling for imbalanced data classification. *International Journal of Knowledge Engineering and Soft Data Paradigms*, 3(1):4–21, 2011. 37

[87] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572, 1901. 7

[88] María Pérez-Ortiz, Peter Tiňo, Rafal Mantiuk, and César Hervás-Martínez. Exploiting synthetically generated data with semi-supervised learning for small and imbalanced datasets. *CoRR*, abs/1903.10022, 2019. 68

[89] Zhenxing Qin, Alan Tao Wang, Chengqi Zhang, and Shichao Zhang. Cost-sensitive classification with k-nearest neighbors. In *International Conference on Knowledge Science, Engineering and Management*, pages 112–131. Springer, 2013. 53

[90] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986. 7, 20, 92

[91] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014. 7, 20, 92

[92] C. J. Van Rijsbergen. *Information Retrieval*. 1979. 48

[93] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958. 7

[94] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019. 90

[95] David E Rumelhart, Geoffrey E Hinton, James L McClelland, et al. A general framework for parallel distributed processing. *Parallel distributed processing: Explorations in the microstructure of cognition*, 1(45-76):26, 1986. 7

[96] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. Citeseer, 1994. 7

[97] Mehdi S. M. Sajjadi, Olivier Bachem, Mario Lucic, Olivier Bousquet, and Sylvain Gelly. Assessing generative models via precision and recall. In *Advances in Neural Information Processing Systems 31*. NeurIPS, 2018. 46

[98] Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990. 7, 25

[99] Jörg Schiller. The impact of insurance fraud detection systems. *Journal of Risk and Insurance*, 73(3):421–438, 2006. 67

[100] Chris Seiffert, Taghi M Khoshgoftaar, Jason Van Hulse, and Amri Napolitano. Rusboost: A hybrid approach to alleviating class imbalance. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 40(1):185–197, 2009. 42

[101] Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948. 20

[102] Shiven Sharma, Colin Bellinger, Bartosz Krawczyk, Osmar R. Zaïane, and Nathalie Japkowicz. Synthetic oversampling with the majority class: A new perspective on handling extreme imbalance. *IEEE International Conference on Data Mining*, pages 447–456, 2018. 68, 78

[103] Harald Steck. Hinge rank loss and the area under the roc curve. In Joost N. Kok, Jacek Koronacki, Raomon Lopez de Mantaras, Stan Matwin, Dunja Mladenič, and Andrzej Skowron, editors, *Machine Learning: ECML 2007*. Springer, 2007. 46

[104] Yanmin Sun, Mohamed S Kamel, Andrew KC Wong, and Yang Wang. Cost-sensitive boosting for classification of imbalanced data. *Pattern recognition*, 40(12):3358–3378, 2007. 41

[105] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. 7

[106] Dacheng Tao, Xiaoou Tang, Xuelong Li, and Xindong Wu. Asymmetric bagging and random subspace for support vector machines-based relevance feedback in image retrieval. *IEEE transactions on pattern analysis and machine intelligence*, 28(7):1088–1099, 2006. 42

[107] Kai Ming Ting. An instance-weighting method to induce cost-sensitive trees. *IEEE Transactions on Knowledge and Data Engineering*, 14(3):659–665, 2002. 41

[108] Ivan Tomek. Two modifications of cnn. *IEEE Transactions on Systems Man and Communications*, 6:769–772, 1976. 34, 46, 90

[109] Peter D Turney. Types of cost in inductive concept learning. *arXiv preprint cs/0212034*, 2002. 40

[110] Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984. 11

[111] V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971. 11

[112] Vladimir N Vapnik. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5):988–999, 1999. 7, 16

[113] R. Viola, R. Emonet, A. Habrard, G. Metzler, S. Riou, and M. Sebban. An adjusted nearest neighbor algorithm maximizing the f-measure from imbalanced data. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 243–250, 2019. 3, 45

[114] Rémi Viola, Rémi Emonet, Amaury Habard, Guillaume Metzler, Sébastien Riou, and Marc Sebban. Une version corrigée de l'algorithme des plus proches voisins pour l'optimisation de la F-mesure dans un contexte déséquilibré. In *Conférence sur l'Apprentissage automatique (CAp 2019)*, Toulouse, France, July 2019. 4, 45

[115] Rémi Viola, Rémi Emonet, Amaury Habrard, Guillaume Metzler, and Marc Sebban. MLFP: Un algorithme d'apprentissage de métrique pour la classification de données déséquilibrées. In *Conférence sur l'Apprentissage automatique (CAp 2020)*, Vannes, France, June 2020. 4, 67

[116] Rémi Viola, Léo Gautheron, Amaury Habrard, and Marc Sebban. Metaap: a meta-tree-based ranking algorithm optimizing the average precision from imbalanced data. *Pattern Recognition Letters*, revised version under review, 2022. 3, 89

[117] Rémi Viola, Rémi Emonet, Amaury Habrard, Guillaume Metzler, Sébastien Riou, and Marc Sebban. A nearest neighbor algorithm for imbalanced classification. *International Journal on Artificial Intelligence Tools*, 30(03):2150013, 2021. 3, 45

[118] Rémi Viola, Rémi Emonet, Amaury Habrard, Guillaume Metzler, and Marc Sebban. Learning from few positives: a provably accurate metric learning algorithm to deal with imbalanced data. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, (IJCAI-20)*, pages 2155–2161. International Joint Conferences on Artificial Intelligence Organization, 7 2020. Main track. 3, 61, 67

[119] Nan Wang, Xibin Zhao, Yu Jiang, Yue Gao, and KLISS BNRist. Iterative metric learning for imbalance data classification. In *27th International Joint Conference on Artificial Intelligence*, pages 2805–2811, 2018. 68, 69, 75, 76, 90

[120] Shuo Wang and Xin Yao. Diversity analysis on imbalanced data sets by using ensemble models. In *2009 IEEE symposium on computational intelligence and data mining*, pages 324–331. IEEE, 2009. 42

[121] Yiru Wang, Weihao Gan, Jie Yang, Wei Wu, and Junjie Yan. Dynamic curriculum learning for imbalanced data classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5017–5026, 2019. 90

[122] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992. 7

[123] Kilian Q. Weinberger and Lawrence K. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10(Feb):207–244, 2009. 29, 47, 49, 55, 56, 68, 69, 75, 76

[124] Dennis L Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, 3, 1972. 34, 46, 90

[125] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992. 23

[126] Fei Wu, Xiao-Yuan Jing, Shiguang Shan, Wangmeng Zuo, and Jing-Yu Yang. Multiset feature learning for highly imbalanced data classification. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 1583–1589. AAAI Press, 2017. 68

[127] Huan Xu and Shie Mannor. Robustness and generalization. *Machine learning*, 86(3):391–423, 2012. 11

[128] Xulei Yang, Qing Song, and Yue Wang. A weighted support vector machine for data classification. *International Journal of Pattern Recognition and Artificial Intelligence*, 21(05):961–976, 2007. 40

[129] Pourya Zadeh, Reshad Hosseini, and Suvrit Sra. Geometric mean metric learning. 07 2016. 69, 76

[130] Valentina Zantedeschi, Rémi Emonet, and Marc Sebban. Metric learning as convex combinations of local models with generalization guarantees. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1478–1486, 2016. 78

[131] Shichao Zhang. Cost-sensitive knn classification. *Neurocomputing*, 391:234–242, 2019. 53

[132] Xiaogang Zhang, Dingxiang Wang, Yicong Zhou, Hua Chen, Fanyong Cheng, and Min Liu. Kernel modified optimal margin distribution machine for imbalanced data classification. *Pattern Recognition Letters*, 125:325–332, 2019. 68

[133] Xiuzhen Zhang and Yuxuan Li. A positive-biased nearest neighbour algorithm for imbalanced classification. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 293–304. Springer, 2013. 49

[134] Xiuzhen Zhang, Yuxuan Li, Ramamohanarao Kotagiri, Lifang Wu, Zahir Tari, and Mohamed Cheriet. Krnn: k rare-class nearest neighbour classification. *Pattern Recognition*, 62:33 – 44, 2017. 49, 55, 56

Abstract Among the dozen missions of the General Direction of Public Finance, there are the calculation of taxes, the control of tax returns and the collection of public revenues. This then allows for the control and execution of public investments. However, as it is often the case in this situation, given the financial aspects involved, some taxpayers, whether individuals or professionals, try to avoid this obligation to declare their income or pay their taxes. In order to solve this problem, the DGFiP has created a service to program the controls using, among others things, Machine Learning. This field of Artificial Intelligence consists in developing algorithms, which will allow a computer to learn, from the collected data, to automate non-trivial tasks initially done by humans. It will then perform them as well or better and in a shorter time. The difficulty for the DGFiP lies in the imbalance between the number of interesting cases, the frauds, and the others. This particular context of fraud detection, which is therefore linked to Imbalanced Learning, requires specific techniques because traditional Machine Learning algorithms struggle to find the cases of interest due to their too low number. Among the specific methods, we find notably the optimization of adapted measures to this imbalanced context, resampling, cost sensitive methods weighting the different errors or even ensemble methods using several algorithms. The goal of this thesis is to develop new fraud detection methods to help the DGFiP in its tracking of fraudsters. Our first contribution is an adaptation of the k nearest neighbor algorithm to the imbalanced context. We show that by weighting the distance between the considered example and its fraudulent neighbors, we facilitate the detection of new suspicious cases, by simulating an expansion of the zone of influence of the fraudsters. Our second contribution is an extension of the first one, using a Metric Learning approach and providing theoretical guarantees through the Uniform Stability framework. The idea is, once again, to use the learned metric only when comparing the considered example with its fraudulent neighbors, thus simulating, in the representation space induced by the metric, a rapprochement of the suspicious examples towards the studied point. Our third contribution concerns an algorithm for producing interpretable rankings. By creating Meta-Decision Trees optimizing the Average Precision at each step of the process, the algorithm addresses one of the problems of the DGFiP which has a limited number of controllers available: maximizing the number of suspicious cases at the top of the ranking and obtaining a decision rule that can be easily interpreted by the end user who will perform the control.

Résumé Parmi la douzaine de missions de la Direction Générale des Finances Publiques, il y a le calcul des impôts, le contrôle des déclarations fiscales et le recouvrement des recettes publiques. Ceci permettant ensuite de contrôler et d'exécuter les dépenses publiques. Cependant, comme souvent dans ces cas là, compte tenu des aspects financiers mis en jeu, certains contribuables, qu'ils soient particuliers ou professionnels, tentent de se soustraire à cette obligation de déclarer leurs revenus ou de payer leurs impôts. Afin de pallier ce problème, la DGFiP a mis en place un service de programmation des contrôles se servant entre autre de l'Apprentissage Automatique. Ce champ de l'Intelligence Artificielle consiste à élaborer des algorithmes, qui vont permettre à un ordinateur d'apprendre, à partir des données récoltées, à automatiser des tâches non triviales et initialement faites par l'homme. Il les fera alors aussi bien voire mieux et en un temps souvent moins long. La difficulté pour la DGFiP réside dans le déséquilibre entre le nombre de cas intéressants, les fraudeurs, et le nombre des non-fraudeurs. Ce contexte particulier de la détection de fraudes, que l'on rattache donc à l'Apprentissage Déséquilibré, nécessite en effet des techniques spécifiques car les algorithmes traditionnels peinent à retrouver les cas d'intérêts du fait de leur trop faible nombre. Parmi les méthodes spécifiques, on retrouve notamment l'optimisation de mesures adaptées à ce contexte déséquilibré, le ré-échantillonnage, les méthodes sensibles aux coûts, pondérant les différentes erreurs, ou encore les méthodes ensemblistes, utilisant plusieurs algorithmes. Le but de cette thèse est de développer de nouvelles méthodes de détection de fraudes afin d'aider la DGFiP dans sa traque des fraudeurs. Notre première contribution est une adaptation de l'algorithme des k plus proches voisins au contexte déséquilibré. Nous montrons qu'en pondérant la distance entre un exemple à étudier et ses voisins fraudeurs, nous facilitons la détection des nouveaux cas suspects, en simulant une expansion de la zone d'influence des fraudeurs. Notre seconde contribution se veut être un prolongement de la première, utilisant une approche d'Apprentissage de Métrique et apportant des garanties théoriques grâce au cadre de Stabilité Uniforme. L'idée est, encore une fois, de n'utiliser la métrique apprise que lors de la comparaison d'un exemple à étudier avec ses voisins fraudeurs, simulant ainsi, dans l'espace de représentation induit par la métrique, un rapprochement des exemples suspects vers le point étudié. Notre troisième contribution porte quant à elle sur un algorithme de génération de classement interprétable. En créant des Méta-Arbres de Décision optimisant l'Average Precision à chaque étape du processus, l'algorithme répond à une des problématiques de la DGFiP qui a un nombre limité de contrôleurs disponibles: maximiser le nombre de cas suspects dans le haut du classement et obtenir une règle de décision facilement interprétable par l'utilisateur final qui fera le contrôle.