

## Introduction à l'apprentissage automatique

Frédéric SUR

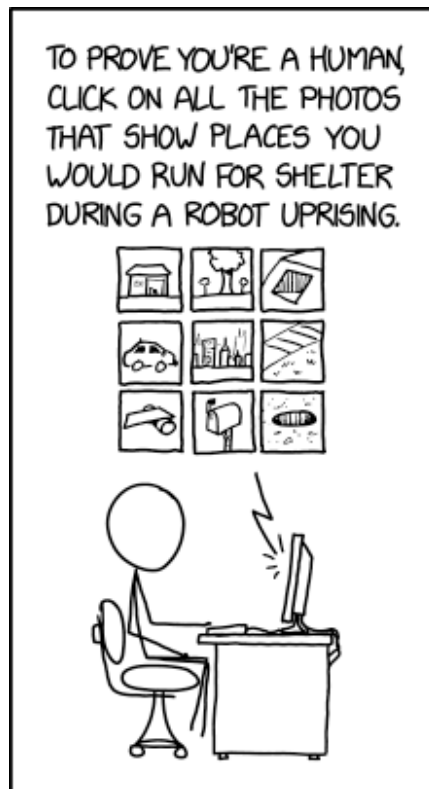
frederic.sur@univ-lorraine.fr

<https://members.loria.fr/FSur/>

Tronc commun scientifique FICM 2A

École des Mines de Nancy

2024-2025



*Machine Learning Captcha*

<https://xkcd.com/2228/>



# Table des matières

|   |           |
|---|-----------|
| <b>À l'attention des étudiants FICM 2A</b>                        | <b>5</b>  |
| <b>1 Introduction</b>   | <b>9</b>  |
| 1.1 Qu'est-ce que l'apprentissage automatique? . . . . .          | 9         |
| 1.2 Les données . . . . .   | 10        |
| 1.3 Apprentissage non-supervisé . . . . .                         | 11        |
| 1.4 Apprentissage supervisé . . . . .                             | 13        |
| 1.5 Pour approfondir. . . . .                                     | 21        |
| <b>2 Deux limites fondamentales de l'apprentissage</b>            | <b>23</b> |
| 2.1 La malédiction de la dimension . . . . .                      | 23        |
| 2.2 Dilemme biais-fluctuation . . . . .                           | 28        |
| 2.3 Pour approfondir. . . . .                                     | 34        |
| <b>3 Problèmes de partitionnement</b>                             | <b>39</b> |
| 3.1 Méthodes hiérarchiques . . . . .                              | 39        |
| 3.2 Partitionnement en $K$ -moyennes . . . . .                    | 42        |
| 3.3 Méthodes de partitionnement basées sur la densité . . . . .   | 48        |
| 3.4 Pour approfondir. . . . .                                     | 52        |
| <b>4 Théorie statistique de la décision</b>                       | <b>55</b> |
| 4.1 Minimisation du risque de prédiction . . . . .                | 55        |
| 4.2 Pour approfondir. . . . .                                     | 63        |
| <b>5 Estimation de densités de probabilité</b>                    | <b>65</b> |
| 5.1 Méthodes non-paramétriques . . . . .                          | 65        |
| 5.2 Méthodes paramétriques . . . . .                              | 68        |
| 5.3 Le retour de la malédiction de la dimension . . . . .         | 73        |
| 5.4 Pour approfondir. . . . .                                     | 75        |
| <b>6 Mise en œuvre du classifieur de Bayes</b>                    | <b>77</b> |
| 6.1 Classifieur naïf de Bayes . . . . .                           | 77        |
| 6.2 Régression logistique . . . . .                               | 79        |
| 6.3 Classification au sens des $P$ plus proches voisins . . . . . | 82        |
| 6.4 Résumé : mise en œuvre du classifieur de Bayes . . . . .      | 84        |
| 6.5 Pour approfondir. . . . .                                     | 84        |

|   |            |
|---|------------|
| <b>7 Méthodes ensemblistes : <i>bagging</i> et <i>boosting</i></b>                      | <b>87</b>  |
| 7.1 Méthodes ensemblistes, classifieurs faibles et forts . . . . .                      | 87         |
| 7.2 Techniques de <i>bagging</i> . . . . .  | 88         |
| 7.3 Techniques de <i>boosting</i> . . . . .   | 94         |
| 7.4 Pour approfondir. . . . .   | 100        |
| <b>8 Machines à vecteurs supports</b>   | <b>101</b> |
| 8.1 Notion de marge maximale . . . . .  | 101        |
| 8.2 Marge souple et variables d'écart . . . . .   | 105        |
| 8.3 Astuce du noyau . . . . .   | 107        |
| 8.4 Retour sur les classifieurs du cours . . . . .                                      | 114        |
| 8.5 Régression à vecteurs supports . . . . .  | 118        |
| 8.6 Pour approfondir. . . . .   | 120        |
| <b>9 Les réseaux de neurones artificiels</b>  | <b>125</b> |
| 9.1 Le perceptron (neurone artificiel) . . . . .  | 125        |
| 9.2 Perceptron multicouche ou réseau de neurones artificiels . . . . .                  | 131        |
| 9.3 Expressivité des réseaux de neurones artificiels . . . . .                          | 137        |
| 9.4 Apprentissage et rétropropagation . . . . .   | 139        |
| 9.5 Problème du sur-apprentissage . . . . .   | 145        |
| 9.6 Conclusion sur les réseaux de neurones pré-2012. . . . .                            | 146        |
| 9.7 Pour approfondir. . . . .   | 147        |
| <b>10 Introduction aux réseaux de neurones convolutifs et à l'apprentissage profond</b> | <b>149</b> |
| 10.1 Le retour des réseaux de neurones . . . . .  | 149        |
| 10.2 Réseaux de neurones convolutifs . . . . .  | 150        |
| 10.3 L'exemple de VGG16 . . . . .   | 156        |
| 10.4 Apprentissage par transfert et réglage fin . . . . .                               | 157        |
| 10.5 Pour approfondir. . . . .  | 158        |
| <b>A Quelques résultats utiles</b>  | <b>161</b> |
| A.1 Inégalités de Hoeffding . . . . .   | 161        |
| A.2 Dérivation des fonctions composées . . . . .  | 161        |
| A.3 Matrices symétriques, classification des quadriques et coniques . . . . .           | 162        |
| <b>B Rappels d'optimisation</b>   | <b>167</b> |
| B.1 Éléments d'optimisation convexe . . . . .   | 167        |
| B.2 Dualité de Wolfe . . . . .  | 169        |
| B.3 Optimisation numérique par algorithme de descente . . . . .                         | 170        |
| <b>Index</b>  | <b>175</b> |

# À l'attention des étudiants FICM 2A

## *(à lire attentivement avant la première séance)*

**Objectifs pédagogiques** Le but de ce cours *Introduction à l'apprentissage automatique* du tronc commun scientifique 2A de Mines Nancy est de permettre la compréhension des enjeux scientifiques et techniques de l'intelligence artificielle, ainsi que la mise en œuvre pratique d'algorithmes d'apprentissage. Le volume horaire disponible étant limité, le panorama que nous dresserons sera représentatif mais nécessairement partiel. Nous aborderons des questions théoriques permettant de comprendre les limites fondamentales de l'apprentissage, l'exposition de quelques modèles, des éléments de méthodologie, et la pratique dans un environnement de programmation très utilisé en milieu académique et industriel (carnets Jupyter et bibliothèque Python scikit-learn).

Le cours s'adresse à tous les élèves FICM, quelque soit leur département. Il ne s'agit pas d'un cours de mathématiques ou d'informatique. L'apprentissage automatique est à présent utilisé dans de nombreux domaines d'application et tout ingénieur sera, au minimum, appelé à interagir avec des spécialistes, voire à décider du déploiement d'une « solution IA ». Bon nombre d'entre vous approfondirez le sujet dans des cours spécialisés ou même dans des M2 dédiés, et consoliderez les aspects mathématiques, informatiques, ou applicatifs selon votre domaine de prédilection.

Si vous souhaitez en savoir plus sur la genèse de ce cours, vous pouvez consulter (disponible sur Arche) :

F. Sur, *Enseignement de l'apprentissage automatique à destination d'ingénieur-e-s généralistes*, Academic Journal of Civil Engineering, vol. 41, no. 2, p. 100-105, 2023.

**Polycopié** Ce document expose les aspects fondamentaux de l'apprentissage automatique que vous devrez étudier en autonomie avant de participer au cours magistral et aux travaux pratiques.

Le polycopié est écrit dans l'optique d'être utile à tous. À ce titre, il couvre volontairement plus de sujets que ce qui sera discuté en cours ou évalué au test. Il contient des démonstrations, des approfondissements, ou des suggestions de lectures complémentaires qui pourront servir dans le cadre d'un projet 2A ou 3A, parcours recherche, cours de M2, ou, plus simplement, susciteront la curiosité. Pour faciliter la lecture, les paragraphes facultatifs sont signalés par un liseré grisé comme dans la marge ci-contre. **Les chapitres non traités ou fa-**

**cultatifs ne figurent pas dans la version imprimée par souci d'économie. Celle-ci présente des marges réduites dans le même objectif.** La version intégrale du polycopié est téléchargeable sur la page Arche ainsi qu'à l'URL suivante :

[https://members.loria.fr/FSur/enseignement/apprauto/poly\\_apprauto\\_FSur.pdf](https://members.loria.fr/FSur/enseignement/apprauto/poly_apprauto_FSur.pdf)

Certaines figures sont plus lisibles en couleurs dans le document pdf en ligne.

Vous vous rendrez compte que le polycopié ne se suffit pas à lui-même : les exemples vus en cours et les travaux pratiques facilitent souvent la compréhension. N'hésitez pas à revenir au polycopié après la séance de cours.

Pour faciliter la lecture de ces ouvrages et d'autres ressources, le vocabulaire de la littérature anglo-saxonne est indiqué en italique tout au long du polycopié.

Chaque chapitre se conclut par des suggestions de lecture. Les articles peuvent être trouvés sur votre moteur de recherche préféré. Utilisez également [scholar.google.fr](https://scholar.google.fr) dédié à la littérature académique. N'hésitez pas non plus à faire des recherches sur les concepts simplement évoqués dans le document ou que vous jugerez nébuleux : de nombreuses ressources pertinentes sont disponibles en ligne et il est toujours intéressant de voir différentes présentations du même sujet.

Je souhaite remercier les collègues s'étant succédé dans l'équipe enseignante pour leur relecture attentive et les suggestions d'ajouts ou de clarifications.

**Page Arche du cours** Le calendrier, les supports de cours, les sujets de TP et leur correction, ainsi que les passages à lire en prévision de chaque séance seront disponibles sur la page Arche du cours. Une heure de lecture attentive et de travail sur les corrections des TP précédents est à prévoir avant chaque séance.

**Bibliographie** Le cours s'appuie essentiellement sur les ouvrages suivants :

1. C. Bishop, *Pattern recognition and machine learning*, Springer, 2006
2. T. Hastie, R. Tibshirani, J. Friedman, *The elements of statistical learning*, 2nd edition, Springer 2008.
3. B. Efron and T. Hastie, *Computer age statistical inference*, Cambridge Univ. Press, 2016
4. I. Goodfellow, Y. Bengio, A. Courville, *Deep learning*, MIT Press, 2016.
5. C. Bishop, H. Bishop, *Deep learning : Foundations and concepts*, Springer, 2024.

Ils sont disponibles à la médiathèque et font partie des documents à consulter si vous souhaitez approfondir les sujets traités : <http://bu.univ-lorraine.fr/>.

Le document suivant est un aide-mémoire très utile de résultats mathématiques de premier cycle ou de cours de tronc commun de première année à Mines Nancy :

- G. Thomas, *Mathematics for machine learning*, Univ. of California at Berkeley, 2018.  
<https://gwthomas.github.io/docs/math4ml.pdf>

**Évaluation** Le cours cherchant à satisfaire des aspirations diverses, vous pouvez légitimement vous demander sur quels éléments portera l'évaluation. L'examen final aura pour objectif de vérifier la compréhension des grands principes de l'apprentissage, des principaux

algorithmes, et du traitement de données réelles. Vous trouverez sur Arche des sujets d'examen des années passées.

Frédéric Sur  
5 novembre 2024

(première version de ce document : janvier 2020)

# Notations

Dans ce document, les vecteurs figurent en gras et les matrices en lettres capitales. On identifiera souvent un vecteur et la matrice colonne le représentant.

Voici les principales notations utilisées :

- le produit scalaire euclidien de deux vecteurs  $\mathbf{x}$  et  $\mathbf{y}$  est noté  $\mathbf{x} \cdot \mathbf{y}$ . Rappelons que si les composantes de ces vecteurs sont  $\mathbf{x} = (x^1, x^2, \dots, x^d)$  et  $\mathbf{y} = (y^1, y^2, \dots, y^d)$ , alors  $\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^d x^i y^i$ ;
- la norme euclidienne d'un vecteur  $\mathbf{x}$  est notée  $\|\mathbf{x}\|_2$ . Elle vérifie  $\|\mathbf{x}\|_2^2 = \mathbf{x} \cdot \mathbf{x}$  et pour tous vecteurs  $\mathbf{x}$  et  $\mathbf{y}$  et scalaire  $\lambda \in \mathbb{R}$ ,  $\|\mathbf{x} + \lambda\mathbf{y}\|_2^2 = \|\mathbf{x}\|_2^2 + \|\mathbf{y}\|_2^2 + 2\lambda\mathbf{x} \cdot \mathbf{y}$ ;
- la transposée d'une matrice  $A$  est noté  $A^T$ ;
- le déterminant d'une matrice carrée  $A$  est noté  $|A|$ ;
- l'inverse d'une matrice carrée inversible  $B$  est noté  $B^{-1}$ ;
- le cardinal d'un ensemble fini  $\mathcal{S}$  est noté  $\#\mathcal{S}$ ;
- l'espérance d'une variable aléatoire  $X$  est notée  $E(X)$ ;
- lorsqu'on cherche à optimiser une fonction  $f$ , on notera  $\operatorname{argmin}_{\mathbf{x}} f(\mathbf{x})$  ou  $\operatorname{argmax}_{\mathbf{x}} f(\mathbf{x})$  une valeur de  $\mathbf{x}$  où  $f(\mathbf{x})$  atteint son minimum ou maximum (« la » valeur dans le cas d'un extremum unique).



# Introduction

Ce chapitre introduit le vocabulaire de l'apprentissage automatique (*machine learning* dans la littérature anglo-saxonne). La discipline étant relativement récente et en mutation constante, le vocabulaire évolue et est sujet à des abus de langage, en particulier lorsqu'on francise des termes techniques issus de la littérature scientifique en langue anglaise. L'objectif de cette introduction est également de dresser un panorama de l'apprentissage et d'expliquer l'articulation entre les chapitres du cours.

### 1.1 Qu'est-ce que l'apprentissage automatique ?

La définition de l'apprentissage automatique selon Wikipedia (octobre 2022) est :

« L'apprentissage automatique (en anglais *machine learning*, littéralement « apprentissage machine »), apprentissage artificiel ou apprentissage statistique est un champ d'étude de l'intelligence artificielle qui se fonde sur des approches mathématiques et statistiques pour donner aux ordinateurs la capacité d'« apprendre » à partir de données, c'est-à-dire d'améliorer leurs performances à résoudre des tâches sans être explicitement programmés pour chacune. Plus largement, il concerne la conception, l'analyse, l'optimisation, le développement et l'implémentation de telles méthodes. On parle d'apprentissage *statistique* car l'apprentissage consiste à créer un modèle dont l'erreur *statistique moyenne* est la plus faible possible. »

L'objectif du cours est de donner un sens à cette définition : que signifie « apprendre » à partir de données, ou « ne pas être explicitement programmé » pour résoudre une tâche ?

Voici trois exemples de problèmes relevant de l'apprentissage automatique.

#### **Exemple 1.1**

Supposons que l'on dispose d'une collection d'articles de journaux. Comment identifier des groupes d'articles portant sur un même sujet ?

**Exemple 1.2**

Supposons que l'on dispose d'un certain nombre d'images représentant des chiens, et d'autres représentant des chats. Comment classer automatiquement une nouvelle image dans une des catégories « chien » ou « chat » ?

**Exemple 1.3**

Supposons que l'on dispose d'une base de données regroupant les caractéristiques de logements dans une ville : superficie, quartier, étage, prix, année de construction, nombre d'occupants, montant des frais de chauffage. Comment prédire la facture de chauffage à partir des autres caractéristiques pour un logement qui n'appartiendrait pas à cette base ?

Trois grandes approches relèvent de l'apprentissage automatique : l'apprentissage supervisé, l'apprentissage non-supervisé, et l'apprentissage par renforcement. Bien entendu, cette classification est sujette à discussion, l'apprentissage semi-supervisé ou l'apprentissage faiblement supervisé (par exemple) apparaissant aux interfaces de ces approches. Ce cours traite les deux premiers aspects de l'apprentissage, et pas l'apprentissage par renforcement qui relève d'autres méthodes mathématiques et algorithmiques. Dans l'exemple 1, on cherche à regrouper les articles portant sur un même sujet, sans disposer d'exemples d'articles dont on sait a priori qu'ils portent sur ce sujet, et sans connaître à l'avance les sujets à identifier. On parlera donc de problème d'apprentissage non-supervisé. Dans les exemples 2 et 3, on cherche à prédire une caractéristique qui est soit une catégorie (exemple 2), soit un montant de facture (exemple 3), à partir d'exemples pour lesquels on connaît la valeur de cette caractéristique. Il s'agit de problèmes d'apprentissage supervisé.

Avant de détailler apprentissage supervisé ou non-supervisé, concentrons-nous sur la notion de données.

## 1.2 Les données

Comme le suggère la définition proposée par Wikipedia, les algorithmes de l'apprentissage automatique sont basés sur des données. On parle aussi d'échantillons (*samples*), d'observations, ou d'exemples. Concrètement, cela signifie que le jeu de données (*dataset*) est formé d'un certain nombre d'articles de journaux (exemple 1), d'images de chiens et chats (exemple 2), ou de caractéristiques de logements (exemple 3). Nous noterons la taille du jeu de données  $N$ , chaque observation  $\mathbf{x}_n$  et le jeu de données de  $N$  observations  $(\mathbf{x}_n)_{1 \leq n \leq N}$ .

Deux grandes familles de jeux de données peuvent être utilisées :

- les données étiquetées : chaque observation  $\mathbf{x}_n$  est fournie avec une étiquette (*label*)  $y_n$  ;
- les données non-étiquetées : comme le nom l'indique, aucune étiquette n'est fournie.

Dans l'exemple 1, les données ne sont pas étiquetées (chaque  $\mathbf{x}_n$  représente un article de journal), alors qu'elles le sont dans l'exemple 2 ( $\mathbf{x}_n$  représente une image, et  $y_n = \text{« chien »}$  ou  $y_n = \text{« chat »}$ ) ou dans l'exemple 3 ( $\mathbf{x}_n$  représente les informations superficie, quartier, étage, prix, année de construction, nombre d'habitants, et  $y_n$  est le montant des frais de chauffage). Il est généralement plus facile de constituer un jeu de données non étiquetées qu'un

jeu de données étiquetées. Dans le premier cas, il « suffit » de collecter des données après pré-traitement automatique minimal, alors que dans le second cas une intervention humaine potentiellement coûteuse est souvent nécessaire pour définir les étiquettes.<sup>1</sup>

Les données peuvent être vues comme des points dans un certain espace. Il est souvent nécessaire de comparer ces points, et il est alors bien pratique que l'espace des données soit muni d'une distance. Dans le cas de données décrites dans un espace vectoriel, les normes usuelles  $\|\cdot\|_1$  ou  $\|\cdot\|_2$  (norme euclidienne) font souvent l'affaire. Rappelons que si  $\mathbf{x}$  est un vecteur de  $\mathbb{R}^d$ , de composantes  $(x^1, \dots, x^d)$ , alors

$$\|\mathbf{x}\|_1 = \sum_{i=1}^d |x^i| \quad \text{et} \quad \|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^d |x^i|^2}$$

où  $|x^i|$  désigne la valeur absolue de la  $i$ -ème composante.

Chaque composante  $x^i$  de l'observation  $\mathbf{x}$  est aussi appelée attribut, dimension, caractéristique (*feature*), ou variable.

Dans l'exemple 2, les observations sont des images décrites comme un vecteur en listant la valeur des niveaux de gris en chaque pixel (pour simplifier, on suppose qu'il s'agit d'images noir et blanc). Les caractéristiques sont les niveaux de gris en chaque pixel. La dimension  $d$  peut valoir un million s'il s'agit d'images de taille réaliste, disons  $1000 \times 1000$  pixels. Dans l'exemple 3, chaque observation est composée de 6 caractéristiques, donc  $d = 6$ .

Dans l'exemple 1, les observations ne sont pas naturellement des éléments d'un espace vectoriel, et on peut se demander comment définir une distance entre documents telle que des documents portant sur le même sujet soient proches au sens de cette distance.

On peut noter que le nombre d'observations disponibles peut fortement varier selon l'application : on pourra sans doute extraire un grand nombre d'images de chats et chiens de base de photographies comme Flickr (le 14 octobre 2022, 886 604 photographies sur Flickr portent le label « dog » et 865 421 « cat »), mais il est peu probable de disposer d'un aussi grand nombre de données relatives aux appartements. Comme on l'a vu précédemment, la dimension  $d$  dépend également de l'application.

*Remarque.* Par exception, nous n'écrirons pas en gras les étiquettes  $y_n$  car, en général, elles désigneront un numéro de classe dans le cas de la classification supervisée ou un scalaire dans le cas de la régression, et plus rarement un vecteur.

### 1.3 Apprentissage non-supervisé

L'apprentissage non-supervisé (*unsupervised learning*) traite des données non-étiquetées. L'objectif est d'identifier automatiquement des caractéristiques communes aux observations.

1. Pour prendre l'exemple des *Facebook papers* il y a quelques années : "Internal Facebook documents show some staff expressing skepticism and include evidence that the company's moderation technology is less effective in emerging markets. One reason for that is a shortage of human-labeled content needed to train machine learning algorithms to flag similar content by themselves." Wired, 25 octobre 2021. <https://www.wired.com/story/facebook-global-reach-exceeds-linguistic-grasp/>

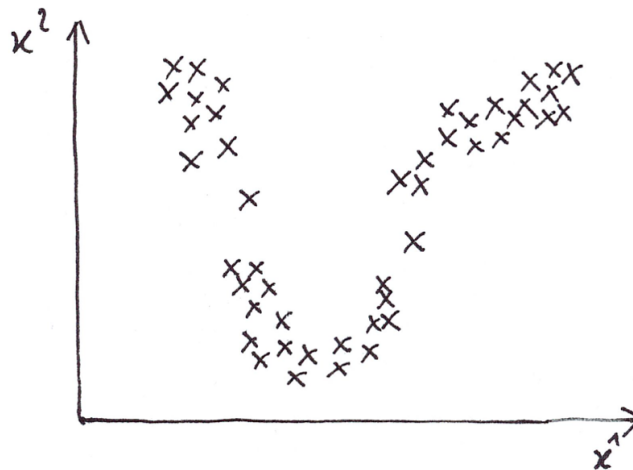


FIGURE 1.1 – *Apprentissage non-supervisé. Ici, les données non-étiquetées sont des points dans un espace bidimensionnel de composantes  $(x^1, x^2)$ . On peut chercher une densité de probabilité ayant permis de générer les observations représentées sur la figure. On peut aussi chercher à identifier des groupes. Dans cet exemple, il semble naturel d'identifier trois groupes. Notons qu'ils ne sont pas nécessairement isotropes (la « forme » d'un groupe n'est pas sphérique), n'ont pas le même nombre d'éléments, et que l'appartenance de certains points à un groupe ou l'autre est ambiguë.*

Les méthodes de réduction de dimension, comme l'analyse en composantes principales, ou les méthodes d'estimation de densités de probabilité font partie de l'apprentissage non-supervisé. Elles sont abordées dans d'autres cours à Mines Nancy. Dans ce cours, nous nous intéresserons essentiellement au problème du partitionnement (*clustering*), dans lequel l'objectif est d'identifier automatiquement des groupes (ou *clusters*) d'observations partageant des profils communs. On peut aussi parler de classification non-supervisée. Intuitivement, les observations groupées ensemble doivent être davantage similaires entre elles qu'elles le sont d'observations d'un autre groupe. Pour mesurer la similarité entre observations, il faut disposer d'une distance  $D$  entre observations. Le problème du partitionnement est illustré par la figure 1.1.

La question qui se pose alors est de savoir comment on identifie les groupes à partir des observations du jeu de données. On comprend également qu'il y a une certaine ambiguïté dans l'objectif du partitionnement : « identifier des groupes d'observations partageant un profil commun ». Un problème important est donc de fixer un seuil sur la distance  $D$  permettant de décider ce qu'est la limite à « partager un profil commun ». De la même manière, on peut se demander quel est le nombre de groupes à identifier dans l'ensemble des observations.

Le **chapitre 3** discute plusieurs approches pour le partitionnement : les classifications hiérarchiques, l'algorithme des  $k$ -moyennes, DBSCAN, et *mean shift*.

L'estimation de densités de probabilité est un autre problème d'apprentissage non-super-

visé. La question à résoudre est : étant donné un ensemble d'observations, quelle distribution de probabilité pourrait l'avoir généré? Nous donnerons des éléments de réponse au **chapitre 5** où nous verrons des méthodes d'estimations non-paramétriques (histogrammes, fenêtres de Parzen) et des méthodes paramétriques (en particulier le mélange de gaussiennes).

## 1.4 Apprentissage supervisé

L'apprentissage supervisé (*supervised learning*) s'intéresse aux données étiquetées. L'objectif est de prédire l'étiquette (inconnue)  $y$  associée à une nouvelle observation  $\mathbf{x}$ , à partir de la connaissance fournie par les  $N$  observations étiquetées du jeu de données  $(\mathbf{x}_n, y_n)_{1 \leq n \leq N}$ .

Pour ce faire, la quasi-totalité des méthodes vues dans ce cours considèrent une fonction de prédiction  $f_{\mathbf{w}}$  appartenant à une famille paramétrée par  $\mathbf{w}$ . Le vecteur  $\mathbf{w}$  représente un ensemble de paramètres. Le nombre de paramètres peut être petit (cas de la régression linéaire) ou très grand (cas de l'apprentissage profond). La prédiction de l'étiquette d'une nouvelle observation  $\mathbf{x}$  ne faisant pas partie du jeu de données sera alors  $f_{\mathbf{w}}(\mathbf{x})$ . Dans une phase d'apprentissage (*learning*), aussi appelée entraînement (*training*), les paramètres  $\mathbf{w}$  sont adaptés de manière à optimiser les performances de prédiction sur le jeu de données  $(\mathbf{x}_n, y_n)_{1 \leq n \leq N}$ , appelé ici base d'apprentissage ou d'entraînement. L'apprentissage est basé sur la mesure de l'écart entre les « vraies » étiquettes  $y_n$  et les étiquettes prédites  $f_{\mathbf{w}}(\mathbf{x}_n)$ . Comme on le verra, il ne suffit pas de choisir  $\mathbf{w}$  minimisant cet écart pour obtenir des performances de prédiction optimales.

Dans un second temps (phase de test), on prédit l'étiquette d'une nouvelle observation  $\mathbf{x}$  comme la valeur de  $f_{\tilde{\mathbf{w}}}(\mathbf{x})$ , à l'aide de l'ensemble de paramètres  $\tilde{\mathbf{w}}$  obtenu par la phase d'apprentissage.

Les problèmes à résoudre sont le choix de la famille à laquelle appartient la fonction de prédiction  $f_{\mathbf{w}}$ , la manière de mesurer l'écart entre vraie étiquette et étiquette prédite, ou le choix du critère d'optimisation de  $\mathbf{w}$ .

On distingue deux grandes familles de problèmes de l'apprentissage supervisé : classification supervisée ( $y$  désigne une classe) et régression ( $y$  est une valeur scalaire ou vectorielle). La distinction est justifiée par les méthodes de résolution sensiblement différentes.

### 1.4.1 Classification supervisée

Lorsque les étiquettes  $y_n$  prennent leurs valeurs dans un ensemble fini dont les éléments correspondent à des catégories (ou classes) à identifier, on parle de classification supervisée. La fonction  $f_{\mathbf{w}}$  associe alors une nouvelle observation  $\mathbf{x}$  à une des classes. Elle définit donc une partition de l'ensemble des observations : chaque élément de cet ensemble sera affecté à une des classes. Les frontières entre classes sont appelées frontières de séparation. L'exemple 2 relève de la classification supervisée, les étiquettes prenant deux valeurs (« chien » ou « chat »). La figure 1.2 montre un exemple de problème de classification supervisée. Sur cette figure, l'observation marquée par « ?? » illustre un dilemme classique : vaut-il mieux la classer avec les ronds, car elle est globalement plus proche des ronds, ou avec les

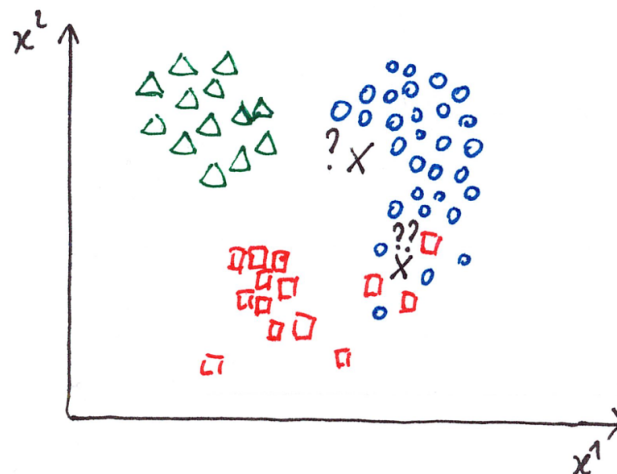


FIGURE 1.2 – *Classification supervisée.* Ici, l'espace des observations est le plan bidimensionnel. Les observations ont des coordonnées  $(x^1, x^2)$  et sont étiquetées par trois catégories (triangle, rond, carré). L'objectif est de déterminer quelle catégorie associer à une nouvelle observation non-étiquetée (représentée par une croix), à partir des observations étiquetées formant la base d'apprentissage. Si, pour l'observation marquée « ? », la tâche peut sembler facile (la classe est vraisemblablement « rond »), la classe de l'observation « ?? » est plus discutable (« rond » ou « carré » ?).

carrés, car elle est très proche de quelques carrés? Autrement dit, la frontière de séparation entre cercles et carrés doit-elle être peu complexe, quitte à mal classer certaines observations de la base d'apprentissage, ou bien très complexe, de manière à séparer finement cercles et carrés et ne pas faire d'erreur de prédiction sur la base d'apprentissage? Ce dilemme est illustré par la figure 1.3; sa discussion sera formalisée au **chapitre 2**.

Le cadre de la théorie statistique de la décision, abordée au **chapitre 4**, permet de définir un classifieur théorique optimal, à savoir le classifieur de Bayes. Néanmoins, ce classifieur reste théorique et ne peut pas être mis en œuvre en pratique sans hypothèses supplémentaires. Selon les hypothèses que l'on impose, le classifieur de Bayes s'incarne en différents modèles de classification supervisée : la classification aux plus proches voisins, le classifieur de la régression logistique (**chapitre 6**), les machines à vecteurs supports (**chapitre 8**), ou les réseaux de neurones artificiels (**chapitre 9**).

Nous verrons aussi comment combiner plusieurs classifieurs dont les performances individuelles sont limitées à l'aide des méthodes ensemblistes au **chapitre 7**.

La classification binaire (ou binaire) s'avère la plus courante : on cherche à classer les observations dans deux classes. Lorsqu'on s'intéresse à  $K > 2$  classes, on peut adapter les classifieurs binaires de la manière suivante :

- Classification « un contre tous » (*one versus all, one against all, ou one versus rest*). Pour chaque classe  $k \in \{1, \dots, K\}$ , on entraîne un classifieur binaire  $f_k$  permettant de discri-

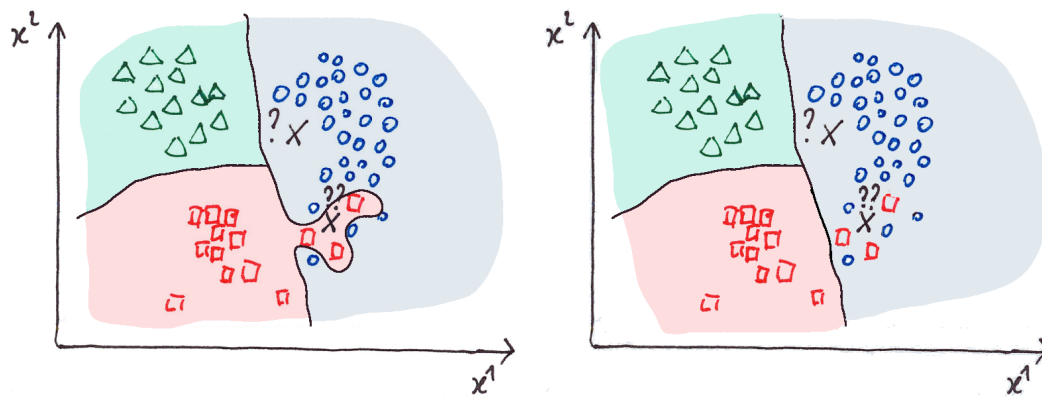


FIGURE 1.3 – *Frontières de séparation sur deux exemples. Quelle frontière de séparation est-elle la plus adaptée : une frontière complexe, permettant de classer correctement toutes les observations de la base d'apprentissage (à gauche), ou une frontière plus régulière, quitte à mal classer certaines de ces observations (à droite) ?*

miner les observations de la classe  $k$  des observations des autres classes, de telle sorte que  $f_k$  prend des valeurs positives sur les observations de la classe  $k$  et négatives sur les autres observations. Ensuite, on affecte une nouvelle observation  $\mathbf{x}$  à  $\arg \max_k f_k(\mathbf{x})$  : la classe dont le classifieur associé a la plus grande valeur. Naturellement, cela suppose que les valeurs prises par les classifieurs soient comparables. D'autre part, tous les classifieurs ne permettent pas de discriminer efficacement deux classes dont l'une (celle regroupant les observations des classes différentes de la  $k$ -ème) est de cardinal beaucoup plus grand que l'autre.

- Classification « un contre un » (*one versus one*). Dans cette approche, on entraîne un classifieur pour chaque paire de classes. Il faut donc entraîner  $K(K-1)/2$  classifieurs. Ensuite, une possibilité est d'affecter une nouvelle observation  $\mathbf{x}$  à la classe majoritaire parmi celles prédites par les  $K(K-1)/2$  classifieurs. Outre le temps de calcul potentiellement grand pour entraîner un tel nombre de classifieurs, il faut gérer les cas d'égalité dans la règle de classification.

Certains classifieurs sont naturellement « multiclassés », comme le classifieur des plus proches voisins, le classifieur naïf de Bayes, ou les réseaux de neurones artificiels. Ils ne nécessitent donc pas d'utiliser une stratégie « un contre tous » ou « un contre un ».

## 1.4.2 Régression

Lorsque les étiquettes  $y_n$  prennent des valeurs scalaires ou vectorielles, on parle de problème de régression. L'exemple 3 est un problème de régression, car on cherche à prédire le montant de la facture de chauffage qui est une grandeur scalaire. La figure 1.4 illustre un problème de régression d'une variable scalaire. On rencontre un dilemme semblable à celui évoqué dans le cas de la classification supervisée. Il est illustré par la figure 1.5.

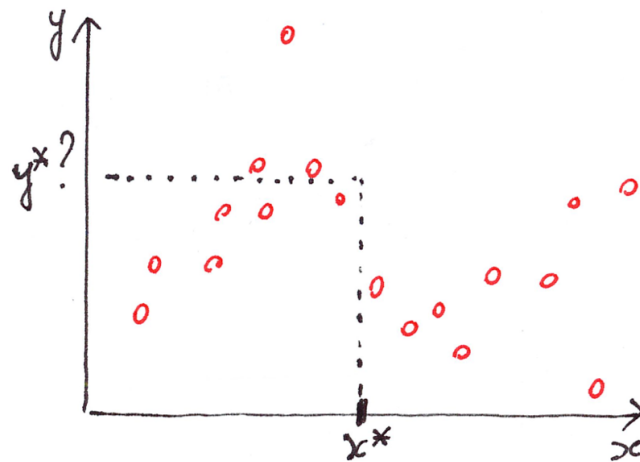


FIGURE 1.4 – Régression. Les observations sont des scalaires  $x$ ; elles sont étiquetées par des scalaires  $y$ , les couples  $(x, y)$  étant représentés dans le plan. L'objectif est de prédire la valeur de l'étiquette  $y^*$  associée à un scalaire  $x^*$ , à partir de la base d'apprentissage  $(x_n, y_n)$  représentée par les ronds. La difficulté est que certaines observations semblent entachées de perturbations aléatoires, certaines observations semblent même aberrantes.

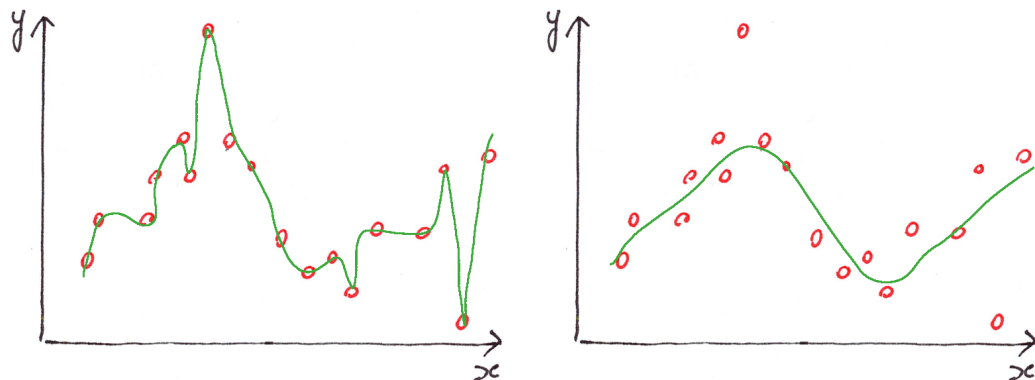


FIGURE 1.5 – Deux exemples de courbes de régression sur la même base d'apprentissage. Vaut-il mieux une courbe passant très près des observations d'apprentissage (à gauche), ou une courbe plus régulière évitant les observations aberrantes et robuste à des perturbations aléatoires des observations, voire robuste à des observations aberrantes (à droite)?

La régression étant abordée dans d'autres cours à Mines Nancy, nous nous concentrerons plutôt sur les problèmes de classification.

Revenons néanmoins brièvement, avec le vocabulaire de l'apprentissage, sur la régression linéaire multivariée que vous connaissez déjà. Dans ce cadre, les étiquettes  $y$  sont scalaires, et les fonctions  $f_w$  sont affines sur l'espace  $\mathbb{R}^d$  des observations. Ces fonctions s'écrivent



sous la forme

$$f_{\mathbf{w}}(\mathbf{x}) = w_0 + \mathbf{w}_1 \cdot \mathbf{x}$$

où  $w_0 \in \mathbb{R}$ ,  $\mathbf{w}_1 \in \mathbb{R}^d$ , et  $\mathbf{w}_1 \cdot \mathbf{x}$  désigne le produit scalaire entre  $\mathbf{w}_1$  et  $\mathbf{x}$ .

### 1.4.2.1 Régression linéaire

Les paramètres  $w_0$  et  $\mathbf{w}_1$  sont obtenus par la méthode des moindres carrés, c'est-à-dire en minimisant la somme des carrés des résidus du modèle sur la base d'apprentissage  $(\mathbf{x}_n, y_n)_{1 \leq n \leq N}$  :

$$\sum_{n=1}^N |y_n - w_0 - \mathbf{w}_1 \cdot \mathbf{x}_n|^2$$

De manière générale, on appelle « résidu » l'écart entre valeur prédite  $f_{\mathbf{w}}(\mathbf{x})$  et « vraie » valeur  $y$ .

Si  $x$  est un scalaire et  $\mathbf{x} = (x, x^2, \dots, x^p)$  est constitué de puissances de  $x$  (jusqu'au degré  $p > 0$ ), on voit que la régression polynomiale apparaît comme un cas particulier de la régression linéaire multivariée. En effet, dans ce cas,  $f_{\mathbf{w}}(\mathbf{x}) = w_0 + \sum_{i=1}^p w_{1,i} x^i$ , où  $\mathbf{w}_1 = (w_{1,1}, \dots, w_{1,p})$ .

Rappelons que les cours de statistique nous donnent l'expression de  $w_0$  et  $\mathbf{w}_1$  en fonction de la base d'apprentissage. Notons  $W$  le vecteur-colonne des paramètres,  $Y$  le vecteur-colonne des étiquettes, et  $X$  la matrice des observations, c'est-à-dire :

$$W = \begin{pmatrix} w_0 \\ \mathbf{w}_1 \end{pmatrix}, \quad Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}, \quad X = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1d} \\ 1 & x_{21} & x_{22} & \dots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & x_{N2} & \dots & x_{Nd} \end{pmatrix}$$

**Proposition 1.1** Avec ces notations, on obtient l'estimation de  $W$  au sens des moindres carrés par les équations dites normales :

$$W = (X^T X)^{-1} X^T Y$$

à la condition que  $X^T X$  soit une matrice inversible.

#### Démonstration

En effet, la quantité à minimiser en fonction des composantes de  $W$  est

$$\sum_{n=1}^N |y_n - w_0 - \mathbf{w}_1 \cdot \mathbf{x}_n|^2 = \|Y - XW\|_2^2$$

et on développe :

$$\|Y - XW\|_2^2 = (Y - XW)^T (Y - XW) = Y^T Y - 2W^T X^T Y + W^T X^T X W$$

Il s'agit d'une fonction quadratique convexe en  $W$ , car  $X^T X$  est symétrique positive (voir annexes B.1 et A.3). L'unique minimum est donc atteint en  $W$  où le gradient s'annule (voir

annexe B.1); on déduit en calculant les dérivées partielles par rapport à chaque composante :

$$-2X^T Y + 2X^T X W = 0$$

d'où les équations normales. □

Les cours de statistique discutent l'effet des observations influentes, qui sont telles que leur variation peut entraîner une forte variation des paramètres  $w_0$  et  $\mathbf{w}_1$ , et des caractéristiques colinéaires (ou fortement corrélées) qui peuvent entraîner la singularité (ou des problèmes de conditionnement) de la matrice  $X^T X$ . Notons également que si le nombre d'observations  $N$  est inférieur à la dimension  $d$ , alors la matrice  $X^T X$  n'est pas inversible. En effet, si  $N \leq d$ , les colonnes de  $X$  sont nécessairement liées.

### 1.4.2.2 Régression ridge

En apprentissage automatique, une approche classique pour surmonter les problèmes d'observations influentes ou de caractéristiques corrélées est de régulariser les paramètres du modèle. Plutôt que minimiser la somme des carrés des résidus, on minimise :

$$\sum_{n=1}^N |y_n - w_0 - \mathbf{w}_1 \cdot \mathbf{x}_n|^2 + \lambda \sum_{j=1}^d |w_j|^2$$

où  $\lambda$  est un paramètre positif ou nul. Il s'agit de la régression *ridge*, dont la régression linéaire est un cas particulier (pour  $\lambda = 0$ ).

Le premier terme est dit « d'attache aux données » car il reflète la capacité du modèle à bien représenter les observations d'apprentissage. Le second terme est dit « de régularisation ».

On démontre que le problème précédent équivaut à minimiser la somme des carrés des résidus, sous contrainte  $\|\mathbf{w}_1\|_2 < C$  où  $C$  est un paramètre positif (il suffit de constater que le Lagrangien du problème d'optimisation sous contrainte est l'expression dépendant du multiplicateur de Lagrange  $\lambda$  ci-dessus). La norme euclidienne de  $\mathbf{w}_1$  qui apparaît dans l'expression à minimiser contraint les paramètres à ne pas prendre de trop grandes valeurs et a pour effet de limiter la dépendance du modèle aux données, d'où l'effet de régularisation.

**Proposition 1.2** *Les paramètres de la régression ridge vérifient :*

$$W = (X^T X + \Lambda)^{-1} X^T Y$$

où la matrice  $\Lambda$  est définie par blocs comme :

$$\Lambda = \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \lambda I_d \end{pmatrix}$$

avec  $I_d$  la matrice identité d'ordre  $d$ .

*Démonstration*

En effet, la fonction à minimiser est :

$$\|Y - XW\|_2^2 + \lambda \|w_1\|_2^2 = Y^T Y - 2W^T X^T Y + W^T X^T X W + W^T \Lambda W$$

qui est convexe car  $\lambda \geq 0$ . En dérivant par rapport aux composantes de  $W$ , on obtient la relation :  $-2X^T Y + 2X^T X W + 2\Lambda W = 0$ , d'où l'expression annoncée.  $\square$

Même si  $X^T X$  n'est pas inversible (dans le cas de caractéristiques colinéaires), la matrice  $X^T X + \Lambda$  le sera pour tout  $\lambda > 0$  car cette matrice est alors symétrique définie positive.

Habituellement,  $w_0$  n'est pas intégré dans le terme de régularisation et n'est donc pas contraint. La raison est que lorsque  $\lambda$  est grand, la régression ridge fournit des  $w_j$  très petit. Lorsque  $w_0$  n'est pas contraint, ce paramètre apparaît alors comme minimisant  $\sum |y_n - w_0|^2$  (les autres termes étant quasiment nuls), donc  $w_0$  est la moyenne des  $y_n$  (voir lemme 1 page 45). Le prédicteur de la régression ridge est alors constant, égal à cette moyenne. Cela fait davantage sens que le prédicteur nul que l'on obtiendrait pour  $\lambda$  grand si on intégrait  $w_0$  dans le terme de régularisation.

Par ailleurs, de manière à ne pas favoriser une caractéristique par rapport à une autre, il vaut mieux que les composantes des observations  $x_n$  varient dans la même gamme de valeurs, quitte à normaliser chaque caractéristique au préalable en retranchant sa moyenne et en divisant par son écart-type.

**1.4.2.3 Le Lasso**

Une autre manière de régulariser le problème de la régression linéaire est de minimiser :

$$\sum_{n=1}^N |y_n - w_0 - w_1 \cdot x_n|^2 + \lambda \sum_{j=1}^d |w_j|$$

qui correspond à la régression Lasso (*Least absolute shrinkage and selection operator*), proposée par Robert Tibshirani en 1996. La seule différence par rapport à la régression ridge est l'utilisation de la norme  $\|\cdot\|_1$  à la place de la norme  $\|\cdot\|_2$  dans le terme de régularisation.

On démontre que le problème précédent est équivalent à minimiser la somme des carrés des résidus, sous contrainte  $\sum_{j=1}^d |w_j| < C$  où  $C$  est un paramètre positif.

L'intérêt d'utiliser la norme  $\|\cdot\|_1$  plutôt que la norme euclidienne est que la résolution du problème de minimisation tendra à annuler certains des  $w_j$ , comme illustré par la figure 1.6. Cela aura pour effet que les caractéristiques correspondantes n'apparaîtront plus dans le modèle : le Lasso permet donc de faire de la sélection de caractéristiques (ou sélection de variables). On parle de modèle parcimonieux (*sparse model*).

Néanmoins, la fonction à minimiser dans le Lasso n'est pas différentiable en  $w = 0$  et il n'y a pas de solution analytique au Lasso : il faut utiliser des algorithmes d'optimisation dédiés que l'on n'étudiera pas dans ce cours.

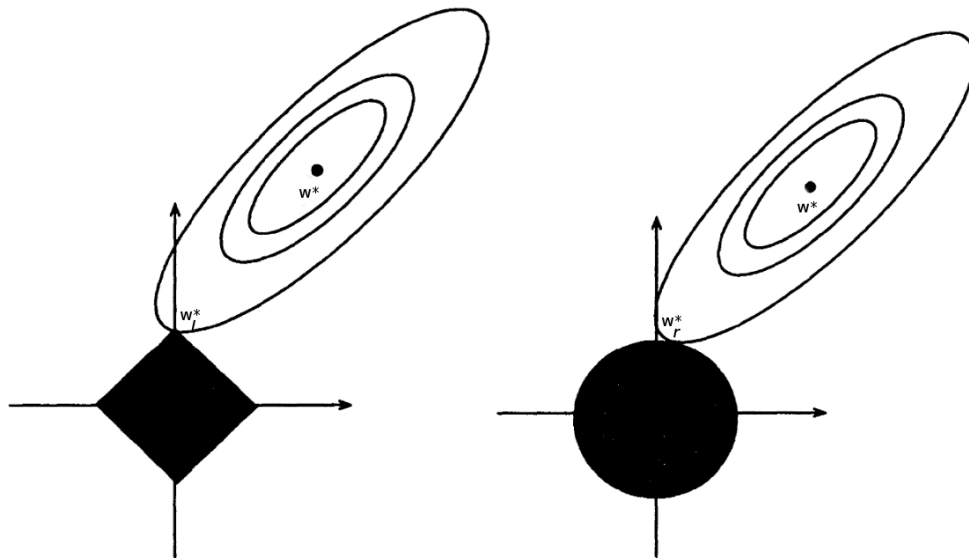


FIGURE 1.6 – Illustration de la minimisation de la somme des carrés des résidus, sous contraintes  $\|\mathbf{w}\|_1 \leq C$  (Lasso, à gauche) ou  $\|\mathbf{w}\|_2 \leq C$  (régression ridge, à droite). L'ensemble des  $\mathbf{w}$  satisfaisant les contraintes forme un carré (cas du Lasso) ou un disque (cas de la régression ridge). La fonction objectif (somme des carrés des résidus) est quadratique et convexe en les composantes de  $\mathbf{w}$ ; ses lignes de niveau sont donc des ellipses centrées en  $\mathbf{w}^*$ , la solution de la régression linéaire classique. La solution  $\mathbf{w}_1^*$  du Lasso ou  $\mathbf{w}_r^*$  de la régression ridge est obtenue respectivement comme point du carré ou du disque appartenant à la ligne de niveau de valeur la plus faible. Il s'agit donc d'un point de la frontière du carré ou du disque lorsque  $\mathbf{w}^*$  ne satisfait pas les contraintes. Sur cet exemple bidimensionnel, on voit que la solution du Lasso a une composante nulle qui n'interviendra donc pas dans le modèle linéaire. Il s'agit d'une propriété générale de la régularisation par norme  $\|\cdot\|_1$ , quelle que soit la dimension des observations : on a de bonne chance pour que l'optimum soit atteint en un point extrémal de l'hypercube unité, où une ou plusieurs composantes sont nulles. Le Lasso sélectionne automatiquement les variables pertinentes dans le modèle et fournit donc des modèles parcimonieux. Illustration adaptée de la figure 2 de l'article Regression Shrinkage and Selection via the Lasso, R. Tibshirani, 1996.

Remarquons que se pose la question de choisir la valeur de  $\lambda$ . Les paramètres comme  $\lambda$  qui sont a priori laissés au choix de l'utilisateur sont appelés « hyperparamètres ». On verra comment fixer les hyperparamètres en pratique, à l'aide de la validation croisée, au **chapitre 2**.

## 1.5 Pour approfondir...

**Intelligence artificielle et apprentissage automatique** Dans ce cours, nous n'employons pas l'expression « intelligence artificielle », qui est un concept très mal défini n'ayant sans doute pas grand-chose à voir avec l'intelligence, si tant est qu'on puisse s'accorder sur une définition de l'intelligence. Toutefois, on peut se demander si un programme informatique aurait un comportement indiscernable d'un comportement humain, ce qui pourrait donner l'illusion d'une certaine forme d'intelligence. C'est l'objet du « jeu de l'imitation » (*imitation game*, d'où le titre du film de 2014) proposé par Alan Turing, souvent appelé test de Turing. On peut remarquer qu'Alan Turing a proposé son test, et a d'ailleurs effectué l'essentiel de ses travaux, à une époque où les ordinateurs n'existaient pas vraiment. Le texte suivant discute de manière didactique le test de Turing, des variantes, et leurs limites :

J.-P. Delahaye, *L'intelligence artificielle et le test de Turing*, Les Nouvelles d'Archimède, No 66, p. 4-6, Université de Lille, 2014

<https://members.loria.fr/FSur/enseignement/apprauto/delahaye14intelligence.pdf>

Le livre suivant présente un panorama assez complet des problématiques scientifiques que l'on peut étiqueter comme relevant de l'intelligence artificielle :

S. Russel, P. Norvig, *Artificial Intelligence : A Modern Approach*, Prentice Hall, 3rd edition, 2009.

Chaque chapitre pourrait faire l'objet d'un cours complet. L'intérêt de ce (gros) livre est de montrer la variété des problématiques de l'IA, mais il peut difficilement être utilisé seul car il ne fait qu'effleurer chaque sujet. Bon nombre de sujets du livre font l'objet de cours à Mines Nancy : recherche opérationnelle, théorie des jeux, logique, représentation de connaissances, inférence statistique, algorithmique, apprentissage, vision par ordinateur, robotique, etc.

**Apprentissage par renforcement** Les élèves intéressé-e-s par l'apprentissage par renforcement pourront consulter :

R.S. Sutton, A.G. Barto, *Reinforcement learning : an introduction*, 2nd edition, MIT Press, 2018.

**Régression ridge et Lasso** Concernant les régressions ridge et Lasso, l'article original de Tibshirani peut être lu (au moins l'introduction) :

R. Tibshirani, *Regression Shrinkage and Selection via the Lasso*, Journal of the Royal Statistical Society Series B (Methodological), Vol. 58, No. 1, p. 267-288, 1996.



# Deux limites fondamentales de l'apprentissage

Nous explorons à présent deux limites de l'apprentissage automatique de nature différente. La première est la « malédiction de la dimension », liée à des propriétés contre-intuitives que peuvent présenter les espaces vectoriels de grande dimension. La seconde concerne le dilemme biais-fluctuation, intrinsèque à toute procédure d'apprentissage.

## 2.1 La malédiction de la dimension

L'expression « malédiction de la dimension » ou « fléau de la dimension » (*curse of dimensionality*) recouvre différents problèmes rencontrés en apprentissage, tous relatifs aux propriétés des espaces de grande dimension qui vont à l'encontre de l'intuition que l'on peut se faire en dimension deux ou trois. L'expression, inventée par le chercheur américain Richard Bellman dans les années 1950-1960, englobe différents phénomènes pas forcément très bien formalisés. Nous allons décrire quelques exemples.

### 2.1.1 Explosion combinatoire

Bellman a initialement formulé la malédiction de la dimension dans un cadre décisionnel. Supposons qu'une politique de décision soit basée sur  $d$  décisions binaires (oui/non) successives, et qu'on cherche une politique optimisant une fonction de coût, c'est-à-dire la suite de décisions fournissant le coût minimum. La recherche de l'optimum peut nécessiter l'exploration de toutes les alternatives, donc des  $2^d$  politiques de décision possibles. Le nombre de possibilités croît exponentiellement avec  $d$ , rendant cette approche inutilisable lorsque  $d$  devient trop grand.

### 2.1.2 Discrétisation du cube unité

Supposons que l'on partitionne le cube unité de  $\mathbb{R}^d$  en « petits » cubes de côté  $1/n$ . Cela nécessite  $n^d$  petits cubes, ce nombre grandissant de manière exponentielle avec la dimension  $d$ , comme illustré par la figure 2.1. Imaginons que l'on cherche à estimer une distribution

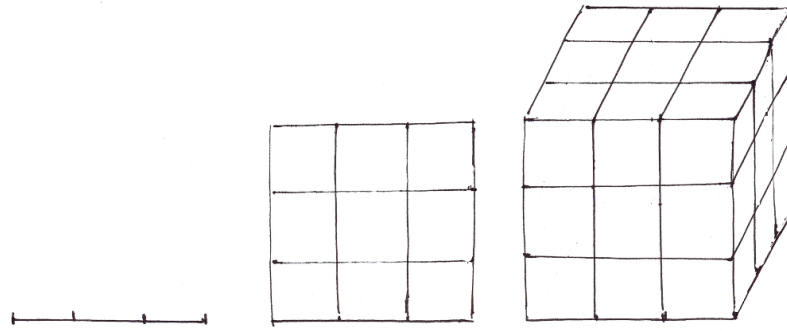


FIGURE 2.1 – Évolution du nombre de cubes de côté  $1/n$  nécessaires pour remplir le cube unité dans  $\mathbb{R}^d$  (ici,  $n = 3$ ).

de probabilité à partir d'un échantillon de 100 points : cela consiste à calculer la proportion de points « tombant » dans chaque petit cube. Si  $d = 1$  et  $n = 10$ , on peut envisager cette approche, car il n'y a que 10 intervalles (cubes en dimension 1) à considérer. Pour obtenir la même finesse de discrétisation ( $1/10$ ) en dimension 10, il faudra considérer  $10^{10}$  soit dix milliards de cubes. L'échantillon devrait être de taille proportionnelle car dans le cas contraire de nombreux cubes seraient vides. On voit qu'estimer une distribution de probabilité en dimension grande est souvent inaccessible sans information additionnelle : cela nécessiterait un échantillon de taille gigantesque.

### 2.1.3 Volume de la boule unité et distance dans $\mathbb{R}^d$

On peut démontrer que le volume  $V_d$  de la boule de  $\mathbb{R}^d$  de rayon 1 est donné par la formule :

$$V_d = \frac{\pi^{d/2}}{\Gamma(d/2 + 1)}$$

où  $\Gamma$  est la « fonction Gamma », définie par

$$\forall x > 0, \Gamma(x) = \int_0^{+\infty} t^{x-1} e^{-t} dt$$

En particulier, on trouve  $V_1 = 2$ ,  $V_2 = \pi$ , et  $V_3 = 4/3\pi$ .

La fonction  $\Gamma$  vérifie

$$\forall n \in \mathbb{N}^*, \Gamma(n) = (n-1)!$$

et croît donc très vite. À l'aide de la formule de Stirling ( $\Gamma(x) \sim \sqrt{2\pi} x^{x-1/2} e^{-x}$ ), on obtient un équivalent du volume  $V_d$  en grande dimension :

$$V_d \sim_{d \rightarrow +\infty} \frac{1}{\sqrt{\pi d}} \left( \frac{2\pi e}{d} \right)^{d/2}$$

Le volume de la boule unité tend donc très vite vers 0 quand la dimension augmente. Cette propriété n'est pas nécessairement très intuitive, mais il y a « pire »...



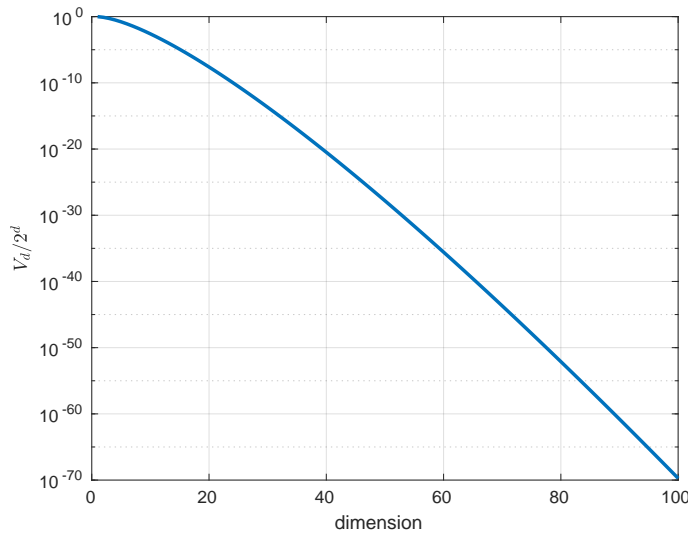


FIGURE 2.2 – Évolution du rapport entre le volume  $V_d$  de la boule unité et le volume  $2^d$  du cube circonscrit en fonction de la dimension  $d$ . Notons l'échelle logarithmique de l'axe des ordonnées.

Considérons un cube de côté 2 circonscrit à la boule unité de  $\mathbb{R}^d$  : son volume est  $2^d$ . La boule unité lui est tangente en chaque face, mais la proportion du volume du cube occupé par la boule est  $V_d/2^d$  et tend donc encore plus vite que  $V_d$  vers 0, comme illustré par la figure 2.2.

Cela signifie que des points répartis aléatoirement de manière uniforme dans ce cube se retrouveront concentrés dans le volume extérieur à la boule unité, puisque le volume de la boule unité devient négligeable relativement à celui du cube. On parle de « concentration dans les coins ».

Par ailleurs, en notant  $V_d^{1-\varepsilon}$  le volume de la boule de rayon  $1 - \varepsilon$  (avec  $0 < \varepsilon < 1$ ), on a la relation suivante :

$$V_d^{1-\varepsilon} = (1 - \varepsilon)^d V_d$$

En effet, cette boule est obtenue de la boule unité par homothétie de rapport  $1 - \varepsilon$ . Ainsi :

$$\frac{V_d - V_d^{1-\varepsilon}}{V_d} = 1 - (1 - \varepsilon)^d$$

Autrement dit, la proportion de la boule unité concentrée dans une couche d'épaisseur  $\varepsilon > 0$  aussi petite que l'on veut, située à la surface de la boule, tend vers 1 lorsque la dimension  $d$  grandit. Ainsi, en grande dimension, tout le volume d'une boule est concentré à sa surface !

Une autre situation paradoxale, lié à ceux présentés ci-dessus, est qu'en grande dimension, toutes les observations sont à la même distance l'une de l'autre. Le graphique de la figure 2.3 illustre l'évolution des distances euclidiennes en grande dimension. Un million de points sont répartis selon la loi aléatoire uniforme dans le cube  $[-1, 1]^d$  de  $\mathbb{R}^d$ , et on calcule,

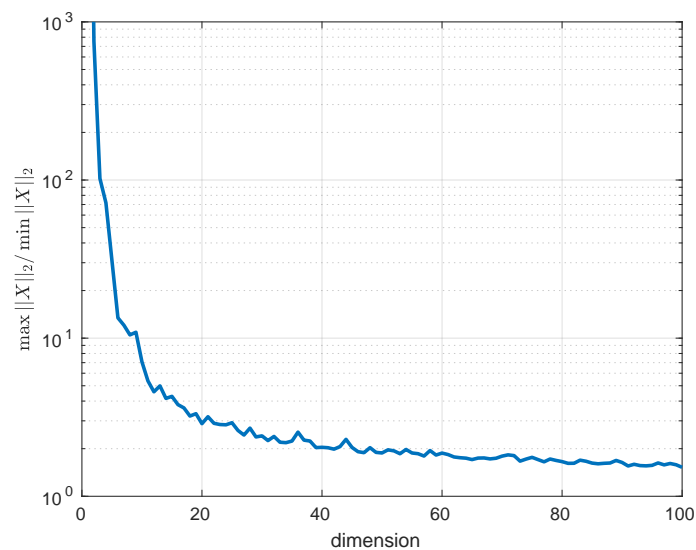


FIGURE 2.3 – Évolution du rapport entre la plus grande distance au centre et la plus petite distance au centre parmi un million de points répartis uniformément dans le cube  $[-1, 1]^d$ , en fonction de la dimension  $d$ .

en fonction de la dimension  $d$ , le rapport de la distance entre le centre du cube et le point le plus proche d'une part, et de la distance entre le centre et le point le plus éloigné d'autre part. On voit que le rapport s'approche de 1 : ces deux distances se rapprochent lorsque la dimension augmente. Il est possible de démontrer sous certaines hypothèses qu'effectivement, leur rapport tend vers 1 lorsque  $d$  tend vers  $+\infty$ . Il s'agit d'une conséquence de la concentration du volume de la boule à sa surface : des points répartis aléatoirement ont plus de chance d'être près de la surface, donc à la même distance au centre de la boule. Ce résultat suggère que la notion de distance perd en pertinence lorsque la dimension est trop grande, puisque tous les points se retrouvent à peu près à la même distance du centre.

On verra dans la suite plusieurs modèles d'apprentissage s'appuyant directement sur la notion de distance : classification au plus proche voisin, partitionnement, estimation de densité, etc. La discussion de cette section suggère que ces modèles ne pourront pas être performants en grande dimension : comment discriminer des observations en fonction de leur distance si toutes les observations sont à peu près à la même distance ?

#### 2.1.4 Phénomène de Hughes

Le phénomène de Hughes indique qu'à base d'apprentissage de taille fixée, le taux d'erreur d'un classifieur diminue avec la dimension des observations (c'est-à-dire le nombre des caractéristiques qui les composent), atteint un minimum, puis croît lorsque la dimension augmente encore.

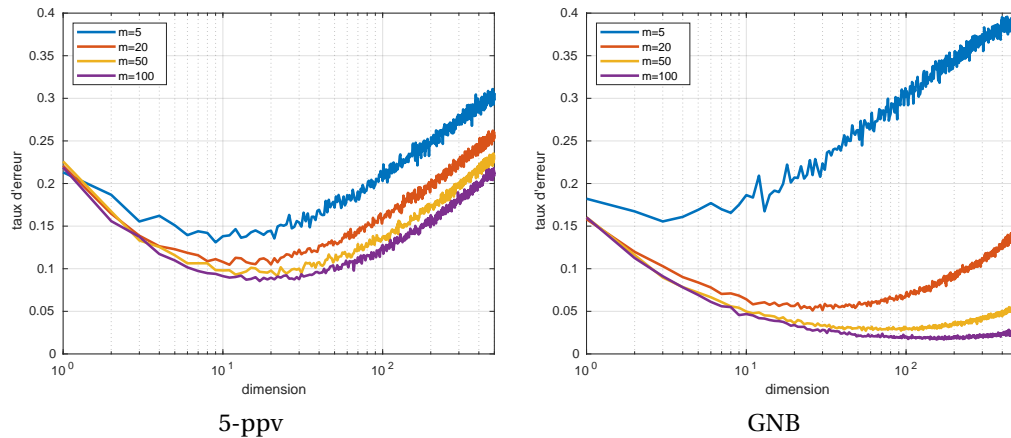


FIGURE 2.4 – Mise en évidence du phénomène de Hughes pour l'expérience de classification décrite dans le texte. À gauche : classifieur aux 5 plus proches voisins. À droite : classifieur de Bayes naïf gaussien. Au delà d'un certain seuil, ajouter des caractéristiques aux observations (et donc augmenter leur dimension) dégrade les performances des classifieurs.

Nous allons illustrer le phénomène sur un exemple numérique<sup>1</sup>. On considère des observations de  $\mathbb{R}^d$  réparties en deux classes :  $m$  observations distribuées aléatoirement selon une loi gaussienne de variance identité centrée en  $\mu_1$  (classe 1), et  $m$  observations distribuées aléatoirement selon une loi gaussienne de variance identité centrée en  $\mu_2 = -\mu_1$  (classe 2). La  $n$ -ème composante de  $\mu_1$  est  $1/\sqrt{n}$ , de sorte que

$$\|\mu_1 - \mu_2\|_2^2 = 4 \sum_{n=1}^d \frac{1}{n}$$

Comme la série harmonique diverge, plus la dimension  $d$  est grande, plus la distance entre les centres  $\mu_1$  et  $\mu_2$  des classes est grande. On pourrait donc penser que le problème de classification devient plus simple lorsque la dimension augmente (car les classes sont davantage « écartées »), et que les performances des classifieurs augmentent.

La figure 2.4 montre le taux d'erreur de deux classifieurs en fonction de la dimension, pour quatre valeurs  $m$  du nombre d'observations dans chaque classe servant à l'apprentissage. À  $m$  et  $d$  fixés, on considère les  $2m$  observations comme base d'apprentissage pour un classifieur (5 plus proches voisins ou classifieur de Bayes naïf gaussien, que l'on verra au chapitre 6). Le taux d'erreur est estimé à partir de 200 observations générées aléatoirement selon les lois précédentes : on calcule simplement la proportion de ces observations correctement classées par le classifieur. Pour limiter les fluctuations d'échantillonnage, on répète 100 fois l'expérience et on représente la moyenne des taux d'erreurs.

Les courbes d'erreur obtenues sont typiques du phénomène de Hughes. Le taux d'erreur des classifieurs augmente lorsque la dimension dépasse un certain seuil dépendant du

1. Exemple inspiré de : G.V. Trunk, *A problem of dimensionality : A simple example*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 1, no. 3, 1979.

nombre d'observations. Le taux d'erreur tend même vers  $1/2$  lorsque la dimension tend vers l'infini : en très grande dimension les classifieurs ne font donc pas mieux qu'un tirage à pile ou face... On peut remarquer dans cette expérience que, selon la taille de la base d'apprentissage et la dimension, l'un ou l'autre des deux classifieurs peut être bien plus performant.

Il s'agit d'une manifestation de la malédiction de la dimension et du fait que les distances ne permettent plus de discriminer les observations lorsque la dimension augmente.

Le phénomène de Hughes est général, on le retrouve dans de nombreuses situations. Ce qu'il faut retenir est qu'essayer de classifier (ou régresser) en grande dimension est rarement une bonne idée, sauf à disposer d'un (potentiellement très) grand nombre d'observations.

### 2.1.5 Quelles solutions pour contrer la malédiction de la dimension ?

Heureusement, si les observations ont  $d$  attributs et sont naturellement plongées dans  $\mathbb{R}^d$ , elles sont souvent incluses dans un hyperplan ou une variété de dimension bien inférieure à  $d$  car les attributs sont généralement liés entre eux.

Si on est capable de décrire les données dans le « bon » espace, on peut se ramener à un espace de dimension bien moins grande qui n'est pas affecté par la malédiction de la dimension, et ce sans perdre d'information utile. L'analyse en composante principale, vue dans un autre cours, est un exemple de technique de réduction de dimension. Parmi les méthodes basées sur les réseaux de neurones, les auto-encodeurs permettent de décrire les observations dans un espace de dimension réduite. Toute une littérature existe sur ce sujet, que l'on n'abordera pas davantage dans ce cours excepté lorsqu'on évoquera le classifieur naïf de Bayes au chapitre 4.

Une autre manière de contrer la malédiction de la dimension est d'utiliser la régularisation. Par exemple, on a vu au chapitre 1 qu'il était impossible de calculer une régression linéaire lorsque le nombre d'observations  $N$  est inférieur à la dimension  $d$  car la matrice  $X^T X$  n'est alors pas inversible. Par contre, on peut toujours calculer la régression ridge car elle nécessite l'inversion de la matrice  $X^T X + \Lambda$ , ce qui est toujours possible lorsque  $\lambda > 0$ .

## 2.2 Dilemme biais-fluctuation

Nous considérons à présent les problèmes d'apprentissage supervisé (régression ou classification), et nous discutons un point évoqué informellement dans le chapitre 1 : a-t-on intérêt à utiliser un modèle de prédiction complexe, se trompant peu sur les données d'apprentissage, ou bien un modèle plus simple quitte à ce qu'il fasse davantage d'erreurs sur les données d'apprentissage ? La discussion sera menée dans un cadre statistique.

### 2.2.1 Notations et hypothèses

On suppose disposer d'un ensemble de  $N$  observations étiquetées  $(\mathbf{x}_n, y_n)_{1 \leq n \leq N}$ , qui feront office de base d'apprentissage. Rappelons que dans le cadre de la régression,  $y_n$  est réel ou vectoriel, et que dans le cadre de la classification,  $y_n$  prend des valeurs dans l'ensemble

fini des catégories possibles. L'objectif est de faire une prédiction  $h(\mathbf{x})$  à partir d'une observation  $\mathbf{x}$  dont on ne connaît pas l'étiquette. On suppose que le prédicteur  $h$  est cherché dans une famille  $\mathcal{H}$  correspondant à un modèle de prédiction. Par exemple, dans le cadre de la régression linéaire multivariée où  $\mathbf{x} \in \mathbb{R}^d$ , on cherche  $h$  dans l'ensemble des fonctions affines :

$$\mathcal{H} = \left\{ h : \mathbf{x} \mapsto w_0 + \mathbf{w}_1 \cdot \mathbf{x}, \text{ tel que } w_0 \in \mathbb{R}, \mathbf{w}_1 \in \mathbb{R}^d \right\}$$

Pour un problème de classification supervisée, on peut par exemple chercher  $h$  parmi les réseaux de neurones à une couche cachée formée de cinq neurones. Dans tous les cas,  $h \in \mathcal{H}$  dépend de paramètres qu'il faudra déterminer en fonction des observations étiquetées.

Le classifieur  $h$  est susceptible de faire des erreurs sur la base d'apprentissage, auquel cas  $h(\mathbf{x}_n) \neq y_n$ . On introduit une fonction  $\ell$  (*loss function*) quantifiant le coût d'une erreur sur l'observation  $(\mathbf{x}_n, y_n) : \ell(y_n, h(\mathbf{x}_n))$ . Par exemple :

— dans le cadre de la régression, on peut choisir

$$\ell(y_n, h(\mathbf{x}_n)) = (y_n - h(\mathbf{x}_n))^2$$

si les  $y_n$  sont scalaires, et

$$\ell(y_n, h(\mathbf{x}_n)) = \|y_n - h(\mathbf{x}_n)\|_2^2$$

si les  $y_n$  sont vectoriels.

L'estimation classique aux moindres carrés revient alors à minimiser  $\sum_{n=1}^N \ell(y_n, h(\mathbf{x}_n))$ .

— dans le cadre de la classification, on peut choisir

$$\ell(y_n, h(\mathbf{x}_n)) = \begin{cases} 0 & \text{si } h(\mathbf{x}_n) = y_n \quad (h \text{ ne fait pas d'erreur sur l'observation } \mathbf{x}_n) \\ 1 & \text{sinon} \end{cases}$$

Il s'agit du coût 0-1 (*0-1 loss*).

Dans les deux cas, on introduit alors le coût moyen des erreurs sur la base d'apprentissage :

$$R_e(h) = \frac{1}{N} \sum_{n=1}^N \ell(y_n, h(\mathbf{x}_n))$$

appelé risque empirique.

Comme le montrent les figures 1.3 et 1.5 du chapitre 1, l'objectif final n'est pas nécessairement de trouver  $h$  minimisant  $R_e$ . Néanmoins,  $R_e$  est la seule mesure d'adéquation entre observations et modèle dont on dispose. L'objectif de cette section est de discuter ce dilemme.

On se place dans le cadre d'une modélisation probabiliste de l'espace des observations étiquetées, que l'on considère comme des réalisations indépendantes d'un couple de variables aléatoires  $(\mathbf{X}, Y)$  de loi de probabilité jointe fixe mais inconnue. Tout ce qu'on connaît sont les  $N$  réalisations  $(\mathbf{x}_n, y_n)_{1 \leq n \leq N}$ . Idéalement, on voudrait utiliser un prédicteur  $h$  qui minimise sur  $\mathcal{H}$  le risque moyen de prédiction, égal à l'espérance du coût d'erreur  $\ell(Y, h(\mathbf{X}))$  sous la loi jointe de  $(\mathbf{X}, Y)$ . En effet, le risque moyen de prédiction représente le coût d'erreur

moyen sur tout l'espace des observations. Il est donc pertinent pour quantifier les performances de  $h$  face à de nouvelles observations. Le risque moyen de prédiction s'écrit formellement comme :

$$R_p(h) = E_{(X,Y)}(\ell(Y, h(\mathbf{X})))$$

où  $E_{(X,Y)}$  désigne l'espérance sous la loi de  $(X, Y)$ .

D'après la loi forte des grands nombres, la base d'apprentissage étant la réalisation d'un échantillon indépendant identiquement distribué suivant la loi de  $(X, Y)$ , le risque empirique  $R_e$  est une bonne approximation du risque de prédiction  $R_p$  lorsque la taille  $N$  de l'échantillon est suffisamment grande. Le problème est que le prédicteur de risque empirique minimal pour chaque  $N$  n'a pas de raison de converger vers le prédicteur de risque moyen de prédiction minimal. Intuitivement, à  $N$  fixé, si le modèle  $\mathcal{H}$  est suffisamment riche on pourra toujours trouver un prédicteur de risque empirique très faible même pour une grande base d'apprentissage, mais de risque moyen de prédiction très grand. Le cas extrême est le prédicteur  $h$  défini par  $h(\mathbf{x}_n) = y_n$  sur la base d'apprentissage, et prenant une valeur quelconque ailleurs ( $h$  apprend donc la base d'apprentissage « par cœur »). Son risque empirique est nul, mais bien évidemment son risque de prédiction va être très élevé. On dit aussi que sa capacité de généralisation est faible.

Notons au passage que si la base d'apprentissage n'est pas représentative des données sur lesquelles on cherche à prédire (par exemple, si on cherche à prédire la facture de chauffage d'appartements parisiens à partir de données d'appartements cairotes), la loi forte des grands nombres ne s'applique pas et le risque empirique est alors un estimateur biaisé du risque de prédiction. Ce qui est développé dans ce chapitre n'aura pas vraiment de sens dans ce cas.

## 2.2.2 Encadrement du risque moyen de prédiction

Notons  $f$  le prédicteur minimisant sur  $\mathcal{H}$  le risque moyen de prédiction  $R_p$ . Idéalement, on voudrait faire nos prédictions avec  $f$  car  $R_p$  mesure le coût des erreurs de prédiction moyen sur toutes les observations étiquetées possibles. Néanmoins,  $R_p$  est inconnu car la loi de probabilité de  $(X, Y)$  est elle-même inconnue, et par conséquent  $f$  ne peut pas être déterminé. On se contente donc de  $\tilde{f}$  dans  $\mathcal{H}$  minimisant le risque empirique  $R_e$  qui est, pour sa part, calculable.

On cherche à quantifier à quel point le risque de prédiction de  $\tilde{f}$  (qui mesure donc sa capacité à généraliser) est éloigné du risque de prédiction minimal  $R_p(f)$ .

On démontre l'encadrement suivant.

**Proposition 2.1** *Si  $f$  minimise le risque moyen de prédiction  $R_p$  et  $\tilde{f}$  minimise le risque empirique  $R_e$  sur  $\mathcal{H}$ , alors :*

$$R_p(f) \leq R_p(\tilde{f}) \leq R_p(f) + 2 \max_{h \in \mathcal{H}} |R_p(h) - R_e(h)|$$

*Démonstration*

Par définition de  $f$  :  $R_p(f) \leq R_p(\tilde{f})$ , d'où la première inégalité.

Par ailleurs :

$$R_p(\tilde{f}) = R_p(\tilde{f}) - R_e(\tilde{f}) + R_e(\tilde{f}) - R_e(f) + R_e(f) - R_p(f) + R_p(f)$$

mais par définition de  $\tilde{f}$  :  $R_e(\tilde{f}) - R_e(f) \leq 0$ , et d'après l'inégalité triangulaire :

$$R_p(\tilde{f}) - R_e(\tilde{f}) + R_e(f) - R_p(f) \leq |R_p(\tilde{f}) - R_e(\tilde{f})| + |R_p(f) - R_e(f)|$$

Donc

$$R_p(\tilde{f}) - R_e(\tilde{f}) + R_e(f) - R_p(f) \leq 2 \max_{h \in \mathcal{H}} |R_p(h) - R_e(h)|$$

D'où la seconde inégalité. □

### 2.2.3 Discussion : sous-apprentissage et sur-apprentissage

La quantité  $R_p(f)$  est l'erreur minimale en dessous de laquelle on ne pourra jamais passer en cherchant un prédicteur dans le modèle  $\mathcal{H}$  : ce terme est dénommé « biais ». Le terme  $\max_{h \in \mathcal{H}} |R_p(h) - R_e(h)|$  mesure les fluctuations de l'écart entre risque moyen et risque empirique sur le modèle  $\mathcal{H}$  : ce terme est donc dénommé « fluctuation ». La proposition 2.1 nous indique que le risque moyen de prédiction de  $\tilde{f}$ ,  $R_p(\tilde{f})$ , n'est pas trop éloigné du risque de prédiction minimal  $R_p(f)$  (le biais) si  $\max_{h \in \mathcal{H}} |R_p(h) - R_e(h)|$  (la fluctuation) a une valeur faible.

Idéalement, on voudrait  $R_p(f)$  faible et  $R_p(\tilde{f})$  pas trop éloigné de  $R_p(f)$ , de manière à assurer une bonne capacité de généralisation du prédicteur  $\tilde{f}$  qui est le seul que l'on peut déterminer.

Néanmoins, le maximum (respectivement le minimum) d'une quantité sur un ensemble grandit (respectivement diminue) lorsque l'ensemble grandit lui-même.

L'inégalité de la proposition 2.1 fait donc apparaître un antagonisme dans le choix du modèle  $\mathcal{H}$ , appelé dilemme biais-fluctuation :

- pour minimiser  $R_p(f)$  (qui, rappelons-le, est le minimum de  $R_p$  sur  $\mathcal{H}$ ), on a intérêt à avoir un modèle  $\mathcal{H}$  riche,
- mais la fluctuation croît avec la richesse du modèle.

D'un point de vue pratique, il faut donc trouver un compromis entre :

- un modèle  $\mathcal{H}$  peu complexe (par exemple avec un ou deux paramètres à estimer) ne pouvant pas bien expliquer les données ( $\mathcal{H}$  trop restrictif), qui donnerait une fluctuation faible mais un biais grand ;
- et un modèle très complexe ( $\mathcal{H}$  gros, avec de nombreux paramètres) fournissant un prédicteur potentiellement bien adapté aux données d'apprentissage ( $R_e(\tilde{f})$  faible), avec un biais  $R_p(f)$  faible mais tel que  $R_p(\tilde{f})$  est très éloigné de l'optimum  $R_p(f)$  (car la fluctuation est grande)

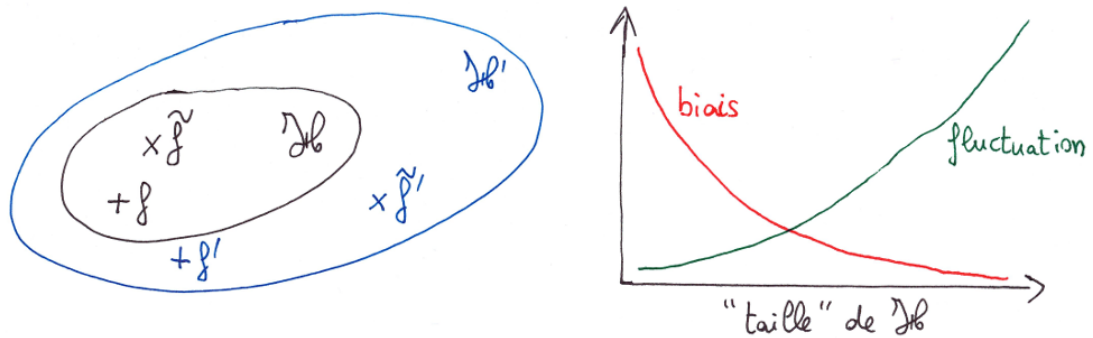


FIGURE 2.5 – Dilemme biais-fluctuation. À gauche : on détermine  $\tilde{f}$  minimisant le risque empirique  $R_e$  sur  $\mathcal{H}$ , que l'on espère pas trop éloigné de  $f$  minimisant le risque moyen de prédiction  $R_p$  sur ce même ensemble. Si on considère  $\mathcal{H}'$  contenant  $\mathcal{H}$ , le minimum de  $R_p$  et  $R_e$  seront respectivement atteints en  $f'$  et  $\tilde{f}'$ , avec des valeurs plus faibles (comme c'est toujours le cas lorsqu'on minimise une fonction sur un domaine élargi). Le biais diminue donc. La contrepartie est que la fluctuation (qui est la valeur du maximum d'une fonction) sera plus grande sur  $\mathcal{H}'$  que sur  $\mathcal{H}$ . À droite : allure de l'évolution du biais et de la fluctuation en fonction de la taille du modèle  $\mathcal{H}$ . Un modèle  $\mathcal{H}$  peu complexe fournira par minimisation du risque empirique un prédicteur  $\tilde{f}$  en sous-apprentissage, et un modèle très complexe un prédicteur  $\tilde{f}$  en sur-apprentissage.

Dans le premier cas, on parle de « sous-apprentissage ». Dans le second, de « sur-apprentissage ». Le dilemme biais-fluctuation est illustré par la figure 2.5. Le graphique de gauche de la figure 1.5 du chapitre 1 montre une situation de sur-apprentissage. Une régression linéaire sur ces données entraînerait un sous-apprentissage.

Par ailleurs, comme le risque empirique  $\frac{1}{N} \sum_{n=1}^N \ell(y_n, h(\mathbf{x}_n))$  est un estimateur sans biais convergent de  $R_p(h)$ , on a intérêt à utiliser une grande base d'apprentissage ( $N$  « grand ») pour diminuer la fluctuation sur un modèle  $\mathcal{H}$ . Cette remarque explique pourquoi l'« intelligence artificielle » est devenue très populaire depuis une dizaine d'années : dans de nombreux domaines applicatifs il est facile d'accéder à de grandes quantités de données, notamment dans le domaine du multimédia. Il suffit de penser à la quantité de données disponibles sur Flickr (créé en 2004), Youtube (2005), Twitter (2006), Facebook (2006)... Cela permet d'utiliser des modèles d'apprentissage de plus en plus complexes. Toutefois, en pratique la base d'apprentissage est généralement fixée : il faut plutôt adapter  $\mathcal{H}$  aux données que l'inverse.

#### 2.2.4 Sélection de modèle et validation croisée

Le prédicteur est choisi de manière à minimiser le risque empirique  $R_e$  dans un modèle  $\mathcal{H}$ . La proposition 2.1 nous dit que le risque moyen  $R_p$  de ce prédicteur optimal est potentiellement éloigné du risque empirique  $R_e$ . La question est de savoir quel modèle choisir pour ne pas se retrouver dans une situation de sous-apprentissage ou de sur-apprentissage.



En pratique, on va réserver une partie (disons 20%) des observations de la base de données pour former une base de validation  $V$  (certains auteurs parlent de base de test), le reste (80%) formant la base d'apprentissage. Le prédicteur  $h$  est choisi en minimisant  $R_e$  calculé sur la base d'apprentissage, et on calcule ensuite le risque empirique sur la base de validation :

$$R_v(h) = \frac{1}{\#V} \sum_{(x_n, y_n) \in V} \ell(y_n, h(x_n))$$

où  $\#V$  est le cardinal de l'ensemble  $V$ .

Comme on l'a vu, le risque empirique  $R_e(h)$  n'est pas nécessairement une bonne approximation du risque moyen de prédiction  $R_p(h)$ . Les observations de la base de validation n'ayant pas servi à l'apprentissage,  $R_v(h)$  est par contre une estimation correcte de  $R_p(h)$ , d'après la loi des grands nombres. Si on reprend l'exemple extrême de l'apprentissage par cœur, le risque empirique sur la base d'apprentissage est nul et ne nous dit rien sur la capacité de prédiction face à de nouvelles observations, alors que le risque empirique sur une base de validation indépendante nous renseigne bien sur cette capacité.

Cela nous conduit à sélectionner un modèle  $\mathcal{H}$  par la procédure suivante.

Pour chaque modèle  $\mathcal{H}$  considéré :

1. on détermine  $\tilde{f}_{\mathcal{H}}$  minimisant  $R_e$  parmi les prédicteurs de  $\mathcal{H}$  ;
2. on calcule  $R_v(\tilde{f}_{\mathcal{H}})$ , valeur du risque empirique sur la base de validation.

Ensuite on sélectionne le modèle  $\mathcal{H}^*$  (et le prédicteur optimal  $\tilde{f}_{\mathcal{H}^*}$ ) minimisant  $R_v(\tilde{f}_{\mathcal{H}})$  parmi les différents modèles  $\mathcal{H}$ .

Notons que le prédicteur  $\tilde{f}_{\mathcal{H}^*}$  sélectionné n'est pas forcément celui qui présente le plus petit risque empirique sur la base d'apprentissage parmi tous les prédicteurs  $\tilde{f}_{\mathcal{H}}$ .

Ce que l'on vient de décrire est la validation *holdout*.

On parle de *learning loss* (ou *training loss*) pour  $R_e$  et de *validation loss* (ou *test loss*) pour  $R_v$ .

L'inconvénient de cette approche est qu'il vaut mieux réserver une part assez importante des données pour l'apprentissage. Le risque calculé sur la base de validation, de taille nécessairement limitée, présente des fluctuations d'échantillonnage potentiellement importantes (en d'autres termes, un autre choix aléatoire de la base de validation  $V$  aurait donné des valeurs différentes pour  $R_v$ ). Ces fluctuations sont gênantes lorsqu'on sélectionne  $\mathcal{H}^*$  car la valeur plus petite de  $R_v(\tilde{f}_{\mathcal{H}^*})$  parmi les  $R_v(\tilde{f}_{\mathcal{H}})$  peut très bien être le résultat du hasard de la construction de la base de validation  $V$  plutôt que des mérites du modèles  $\mathcal{H}^*$ .

Il est possible de moyenniser ces fluctuations en découpant la base de données en  $K$  sous-ensembles, puis en calculant successivement  $R_v$  sur l'un de ces sous-ensembles (qui joue le rôle de base de validation), l'apprentissage étant fait au préalable sur les  $K-1$  sous-ensembles restants. Un score moyen est alors calculé comme la moyenne des  $K$  valeurs de  $R_v$  obtenues. On sélectionne finalement le modèle de score moyen minimal. Il s'agit de la « validation croisée à  $K$  plis » (*K-fold cross validation*). Le cas extrême  $K = N$  correspond à la situation suivante : on entraîne  $N$  fois le prédicteur sur un ensemble de  $N-1$  observations, l'ensemble

| $\mathcal{S}_1$ | $\mathcal{S}_2$ | $\mathcal{S}_3$ | $\mathcal{S}_4$ | $\mathcal{S}_5$ |                               |
|-----------------|-----------------|-----------------|-----------------|-----------------|-------------------------------|
| A               | A               | A               | A               | V               | → coût d'erreur sur V : $R_1$ |
| A               | A               | A               | V               | A               | → coût d'erreur sur V : $R_2$ |
| A               | A               | V               | A               | A               | → coût d'erreur sur V : $R_3$ |
| A               | V               | A               | A               | A               | → coût d'erreur sur V : $R_4$ |
| V               | A               | A               | A               | A               | → coût d'erreur sur V : $R_5$ |

Erreur de validation croisée :  $\frac{1}{5} (R_1 + R_2 + R_3 + R_4 + R_5)$

FIGURE 2.6 – *Validation croisée à cinq plis* : la base des observations est séparée en cinq sous-ensembles (plis)  $\mathcal{S}_1 \dots \mathcal{S}_5$  de taille identique, chaque sous-ensemble est à tour de rôle dans la base d'apprentissage ou fait office de base de validation. L'apprentissage se fait sur les quatre plis d'apprentissage, et un coût d'erreur est calculé sur le pli de validation. Le score de validation croisée est la moyenne des cinq coûts d'erreur obtenus.

de validation  $V$  étant réduit à la  $N^{\text{ème}}$  observation. On parle alors de validation croisée *leave-one-out*. Bien entendu, le temps de calcul peut être assez important car il faut procéder à  $N$  entraînements successifs. En pratique, on se contente souvent d'une validation croisée à  $K = 5$  ou  $K = 10$  plis. La figure 2.6 illustre la validation croisée à cinq plis.

La validation croisée ou la validation *holdout* permettent de choisir un prédicteur pour différents modèles  $\mathcal{H}$ . Souvent, ce qui distingue les modèles  $\mathcal{H}$  sont les valeurs des hyperparamètres. Par exemple, l'hyperparamètre  $\lambda$  dans la régression ridge ou le Lasso du chapitre 1, ou la taille de la couche cachée d'un réseau de neurones. La validation croisée fournit donc une méthode pour fixer les hyperparamètres parmi un ensemble fini de valeurs : on calcule le score de validation croisée des prédicteurs optimaux trouvés pour différentes valeurs (en nombre fini) des hyperparamètres, et on sélectionne le jeu d'hyperparamètres donnant le meilleur score.

## 2.3 Pour approfondir...

**Norme en grande dimension** Le comportement de la norme lorsque la dimension augmente (comme illustré par la figure 2.3) est discuté dans cet article :

G. Biau, D.M. Mason, *High-dimensional  $p$ -norms*, Mathematical Statistics and Limit Theorems, Springer, 2015.

**Réduction de dimension** Pour en savoir plus, les deux articles suivants sont disponibles sur Internet :

C. Burges, *Dimension Reduction : A Guided Tour*, Foundations and Trends in Machine Learning, vol. 2, no. 4, p. 275-365, 2009.

E. Bingham and H. Mannila, *Random projection in dimensionality reduction : applications to*

*image and text data*, proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), p. 245-250, 2001.

**Dilemme biais-variance** L'expression « dilemme biais-variance » est souvent employée dans les cours d'apprentissage. Elle concerne généralement le cadre de la régression avec coût quadratique, et s'exprime de manière légèrement différente que le dilemme biais-fluctuation de la section 2.2, présenté pour sa part dans un cadre plus général.

On suppose toujours disposer d'un ensemble de  $N$  observations étiquetées  $(\mathbf{x}_n, y_n)_{1 \leq n \leq N}$  (un ensemble d'apprentissage), vues comme  $N$  réalisations d'un couple de variables aléatoires  $(X, Y)$ . À présent, on suppose que  $N$  est également réalisation d'une variable aléatoire, en d'autres termes que l'ensemble d'observations étiquetées est réalisation d'un ensemble aléatoire  $\mathcal{E}$ . On considère le prédicteur  $h_{\mathcal{E}}$  estimé à partir d'un tel  $\mathcal{E}$  (par exemple par minimisation du risque empirique sur une famille  $\mathcal{H}$  de prédicteurs) : il peut lui-même être vu comme une variable aléatoire. Pour une fonction de coût  $\ell$  quadratique, le coût d'erreur de  $h_{\mathcal{E}}$  face à une observation étiquetée  $(\mathbf{x}, y)$  inconnue est  $(y - h_{\mathcal{E}}(\mathbf{x}))^2$ . Selon l'ensemble d'observations dont on dispose en pratique, cette erreur peut varier à cause des fluctuations d'échantillonnage : chaque ensemble d'observations fournit un prédicteur différent par apprentissage. On s'intéresse donc à l'erreur de prédiction sur l'observation  $\mathbf{x}$  moyennée sur tous les ensemble d'observations étiquetées possibles. Il s'agit donc de déterminer  $E_{\mathcal{E}}(y - h_{\mathcal{E}}(\mathbf{x}))^2$ . Notons  $\bar{h}(\mathbf{x}) = E_{\mathcal{E}}(h_{\mathcal{E}}(\mathbf{x}))$  : il s'agit de la prédiction moyenne de l'observation  $\mathbf{x}$  sur l'ensemble des observations étiquetées possibles fournissant chacun un prédicteur  $h_{\mathcal{E}}$ .

On calcule successivement :

$$\begin{aligned} E_{\mathcal{E}}((y - h_{\mathcal{E}}(\mathbf{x}))^2) &= E_{\mathcal{E}}\left(\left(y - \bar{h}(\mathbf{x}) + \bar{h}(\mathbf{x}) - h_{\mathcal{E}}(\mathbf{x})\right)^2\right) \\ &= E_{\mathcal{E}}\left(\left(y - \bar{h}(\mathbf{x})\right)^2\right) + E_{\mathcal{E}}\left(\left(h_{\mathcal{E}}(\mathbf{x}) - \bar{h}(\mathbf{x})\right)^2\right) + 2E_{\mathcal{E}}\left(\left(y - \bar{h}(\mathbf{x})\right)\left(h_{\mathcal{E}}(\mathbf{x}) - \bar{h}(\mathbf{x})\right)\right) \end{aligned}$$

Or :

- $E_{\mathcal{E}}\left(\left(y - \bar{h}(\mathbf{x})\right)^2\right) = \left(y - \bar{h}(\mathbf{x})\right)^2$  car la variable aléatoire dont on calcule l'espérance ne dépend pas de  $\mathcal{E}$  ;
- $E_{\mathcal{E}}\left(\left(y - \bar{h}(\mathbf{x})\right)\left(h_{\mathcal{E}}(\mathbf{x}) - \bar{h}(\mathbf{x})\right)\right) = E_{\mathcal{E}}(yh_{\mathcal{E}}(\mathbf{x})) + E_{\mathcal{E}}(\bar{h}(\mathbf{x})^2) - E_{\mathcal{E}}(y\bar{h}(\mathbf{x})) - E_{\mathcal{E}}(h_{\mathcal{E}}(\mathbf{x})\bar{h}(\mathbf{x})) = 0$ . En effet :  $E_{\mathcal{E}}(yh_{\mathcal{E}}(\mathbf{x})) = y\bar{h}(\mathbf{x})$ ,  $E_{\mathcal{E}}(\bar{h}(\mathbf{x})^2) = \bar{h}(\mathbf{x})^2$ ,  $E_{\mathcal{E}}(y\bar{h}(\mathbf{x})) = y\bar{h}(\mathbf{x})$ , et enfin  $E_{\mathcal{E}}(h_{\mathcal{E}}(\mathbf{x})\bar{h}(\mathbf{x})) = \bar{h}(\mathbf{x})^2$ . Les termes s'annulent donc deux à deux.

On conclut. La moyenne sur les ensembles d'apprentissage de l'erreur quadratique de prédiction est :

$$E_{\mathcal{E}}((y - h_{\mathcal{E}}(\mathbf{x}))^2) = \left(y - \bar{h}(\mathbf{x})\right)^2 + E_{\mathcal{E}}\left(\left(h_{\mathcal{E}}(\mathbf{x}) - \bar{h}(\mathbf{x})\right)^2\right)$$

Dans cette décomposition :

- $\left(y - \bar{h}(\mathbf{x})\right)^2$  est le carré de l'écart entre la « vraie » valeur d'étiquette  $y$  et la prédiction moyenne sur les prédicteurs obtenus à partir des ensembles d'observations étiquetées. Il s'agit donc du carré du biais du modèle d'apprentissage.

- $E_{\mathcal{E}} \left( \left( h_{\mathcal{E}}(\mathbf{x}) - \bar{h}(\mathbf{x}) \right)^2 \right)$  mesure la dispersion entre la prédiction de  $h_{\mathcal{E}}$  déterminé sur un ensemble d'apprentissage particulier  $\mathcal{E}$  et la prédiction moyenne. Il s'agit de la variance du modèle d'apprentissage.

Le biais mesure l'écart entre prédiction moyenne et vraie étiquette. En cas de sous-apprentissage, le biais est élevé : le modèle ne capture pas suffisamment d'information et fournit, en moyenne, des prédictions assez éloignées de ce qui est attendu. C'est le cas par exemple si on cherche une régression linéaire alors que les étiquettes  $y$  ne sont pas liées aux  $\mathbf{x}$  de manière linéaire. La variance mesure l'amplitude des changements dans la prédiction si on change l'ensemble d'apprentissage. En cas de sur-apprentissage, la variance est élevée : le modèle s'adapte trop à la base d'apprentissage et varie fortement lorsque la base change.

Un modèle simple sera donc affecté d'un biais élevé mais d'une variance faible (car il n'est pas très affecté par des modifications de la base d'apprentissage), tandis qu'un modèle complexe sera affecté d'un biais faible (avec suffisamment d'observations la prédiction sera correcte) mais d'une variance élevée. Il faut donc réaliser un compromis entre biais et variance.

**Complexité des modèles** La présentation du dilemme biais-fluctuation est adaptée des notes du cours de Stéphane Mallat au Collège de France en 2018 (chaire sciences des données), disponible à cette URL :

<https://www.college-de-france.fr/site/stephane-mallat/course-2017-2018.htm>

Nous avons discuté de manière assez informelle la « complexité » ou la « taille » du modèle  $\mathcal{H}$  qui doit être « pas trop grande » afin de limiter la fluctuation (ou la variance) et donc un potentiel sur-apprentissage. Nous avons souligné que la complexité du modèle devait s'apprécier en fonction du nombre d'observations. En effet, à modèle  $\mathcal{H}$  fixé le risque empirique converge vers le risque de prédiction lorsque le nombre d'observations augmente : nous avons donc intérêt à avoir beaucoup d'observations dans la base d'apprentissage pour limiter la fluctuation (donc le sur-apprentissage). La théorie statistique de l'apprentissage (*statistical learning theory*) fournit des bornes sur la fluctuation qui dépendent de la taille de la base d'apprentissage et de la complexité du modèle, cette complexité étant entendue en un sens précis.

Plaçons-nous dans le cadre de la classification avec coût 0-1, et des observations étiquetées qui sont des réalisations indépendantes identiquement distribuées de  $(\mathbf{X}, Y)$ . Supposons que  $\mathcal{H}$  est constitué d'un nombre fini de classifieurs, son cardinal étant noté  $\#\mathcal{H}$ . Le risque étant borné, l'inégalité de Hoeffding (voir l'annexe A.1) permet d'écrire pour tout classifieur  $h \in \mathcal{H}$  et tout  $\varepsilon > 0$  :

$$\Pr(|R_e(h) - R_p(h)| \geq \varepsilon) \leq 2e^{-2N\varepsilon^2}$$

En effet, on peut utiliser le résultat de l'annexe A.1 avec les notations :  $X_n = \ell(Y_n, h(\mathbf{X}_n))$ ,  $\forall n$ ,  $[a_n, b_n] = [0, 1]$ ,  $\bar{X} = R_e(h)$ , et  $E(\bar{X}) = \frac{1}{N} \sum_{n=1}^N E(\ell(Y_n, h(\mathbf{X}_n))) = R_p(h)$ .

Maintenant, la fluctuation est  $\max_{h \in \mathcal{H}} |R_e(h) - R_p(h)|$ , donc elle est supérieure à  $\varepsilon$  si au

moins un des  $h$  de  $\mathcal{H}$  vérifie l'inégalité précédente. Autrement dit :

$$\begin{aligned} \Pr\left(\max_{h \in \mathcal{H}} |R_e(h) - R_p(h)| \geq \varepsilon\right) &= \Pr\left(\cup_{h \in \mathcal{H}} [|R_e(h) - R_p(h)| \geq \varepsilon]\right) \\ &\leq \sum_{h \in \mathcal{H}} \Pr(|R_e(h) - R_p(h)| \geq \varepsilon) \end{aligned}$$

En effet, la probabilité de l'union finie d'événements est inférieure à la somme des probabilités des événements.

Par conséquent,

$$\Pr\left(\max_{h \in \mathcal{H}} |R_e(h) - R_p(h)| \geq \varepsilon\right) \leq 2\#\mathcal{H} e^{-2N\varepsilon^2}$$

Si on note  $\delta = 2\#\mathcal{H} e^{-2N\varepsilon^2}$ , on obtient

$$\varepsilon = \sqrt{\frac{1}{2N} \log\left(\frac{2\#\mathcal{H}}{\delta}\right)}$$

d'où

$$\Pr\left(\max_{h \in \mathcal{H}} |R_e(h) - R_p(h)| \geq \sqrt{\frac{1}{2N} \log\left(\frac{2\#\mathcal{H}}{\delta}\right)}\right) \leq \delta$$

soit :

$$\Pr\left(\max_{h \in \mathcal{H}} |R_e(h) - R_p(h)| \leq \sqrt{\frac{1}{2N} \log\left(\frac{2\#\mathcal{H}}{\delta}\right)}\right) \geq 1 - \delta$$

Ainsi, avec une probabilité supérieure à  $1 - \delta$ , la fluctuation est bornée par  $\sqrt{\frac{\log(2\#\mathcal{H}) - \log(\delta)}{2N}}$ . On peut noter que cette borne ne dépend pas de la loi (inconnue) de  $(\mathbf{X}, Y)$ .

Comme pour tout  $h \in \mathcal{H}$ ,  $R_p(h) - R_e(h) \leq \max_{h \in \mathcal{H}} |R_e(h) - R_p(h)|$ , on peut conclure par la proposition suivante.

**Proposition 2.2** Si  $\mathcal{H}$  est un modèle de cardinal fini, pour tout  $h \in \mathcal{H}$ , avec probabilité au moins  $1 - \delta$ ,

$$R_p(h) \leq R_e(h) + \sqrt{\frac{\log(2\#\mathcal{H}) - \log(\delta)}{2N}}$$

Avec une probabilité au moins  $1 - \delta$ , le risque moyen de prédiction est majoré par l'expression donnée par la proposition précédente, et on espère que le majorant soit petit.

D'un point de vue pratique, cette borne sur le risque moyen de prédiction fait apparaître un compromis entre risque empirique et  $\log(\#\mathcal{H})/N$ . Minimiser le risque empirique n'assurera un faible risque moyen de prédiction que si le modèle  $\mathcal{H}$  n'est pas trop complexe par rapport à la taille  $N$  de l'ensemble d'apprentissage, la complexité étant mesurée par le cardinal de  $\mathcal{H}$ .

Bien entendu,  $\mathcal{H}$  n'a pas de raison d'être fini, car en général  $\mathcal{H}$  est un ensemble de classificateurs dépendant de paramètres réels. Dans ce cas, des développements théoriques

dans les années 1980-1990 ont permis de mesurer la complexité par la dimension de Vapnik-Chervonenkis ou la complexité de Rademacher. Le problème est que les bornes obtenues sont assez larges : on peut difficilement s'en servir dans un cas pratique pour fixer la taille de l'ensemble d'apprentissage de manière à assurer un risque de prédiction satisfaisant.

Le lecteur souhaitant approfondir le sujet pourra consulter :  
V. Vapnik, *Statistical learning theory*, Wiley, 1998.

# Problèmes de partitionnement

Dans les problèmes de partitionnement, on cherche à identifier automatiquement des groupes (*clusters*) d'observations partageant des profils similaires. Les observations groupées ensemble doivent être davantage similaires entre elles qu'elles le sont d'observations d'un autre groupe. Pour ce faire, on suppose disposer d'une mesure de similarité (ou dissimilarité) entre observations.

Les observations sont non-étiquetées ; il s'agit d'un problème d'apprentissage non-supervisé. Le nombre de groupes à identifier est a priori inconnu, et on ne dispose pas explicitement d'exemples de chaque groupe. Seule la mesure de (dis-)similarité apporte une information : toute méthode de partitionnement essaiera d'arriver à une grande similarité intra-groupes et une faible similarité inter-groupes.

Ce chapitre porte sur les méthodes hiérarchiques de partitionnement, le célèbre algorithme des  $K$ -moyennes, et deux algorithmes basés sur l'estimation de la densité des observations, DBSCAN et *mean shift*. Toutes ces méthodes font usage d'une distance entre observations. Si chaque observation est décrite par un grand nombre de caractéristiques, la malédiction de la dimension risque de frapper : toutes les observations sembleront également semblables (ou également différentes) comme discuté au chapitre 2 (section 2.1.3), empêchant la construction d'un partitionnement cohérent.

## 3.1 Méthodes hiérarchiques

### 3.1.1 Mesures de dissimilarité entre observations et entre groupes

Les méthodes hiérarchiques ascendantes (on parle aussi d'algorithmes de classification hiérarchique ascendante, mais attention, il s'agit bien de méthodes de partitionnement, pas de classification au sens de la Section 1.4.1) nécessitent :

- une mesure de dissimilarité  $d$  entre observations, telle que si  $\mathbf{x}$  et  $\mathbf{y}$  sont deux observations,  $d(\mathbf{x}, \mathbf{y})$  est faible lorsque  $\mathbf{x}$  et  $\mathbf{y}$  sont semblables (et grand dans le cas contraire, d'où l'expression « mesure de dissimilarité ») ;

- une mesure de dissimilarité  $D$  entre groupes d'observations.

Lorsque les observations sont vectorielles, il est possible d'utiliser pour  $d$  les distances induites par les classiques normes  $\|\cdot\|_1$  ou  $\|\cdot\|_2$ . Lorsque les observations sont des mots (c'est-à-dire des chaînes de caractères), on peut utiliser par exemple la distance d'édition, aussi appelée distance de Levenshtein (voir le paragraphe 3.4).

La mesure de dissimilarité  $D$  est généralement construite à partir de  $d$ . Si  $\mathcal{A}$  et  $\mathcal{B}$  sont deux groupes d'observations, on peut définir :

- critère *single linkage* :

$$D_{\text{SL}}(\mathcal{A}, \mathcal{B}) = \min \{d(\mathbf{x}, \mathbf{y}), \mathbf{x} \in \mathcal{A}, \mathbf{y} \in \mathcal{B}\}$$

- critère *complete linkage* :

$$D_{\text{CM}}(\mathcal{A}, \mathcal{B}) = \max \{d(\mathbf{x}, \mathbf{y}), \mathbf{x} \in \mathcal{A}, \mathbf{y} \in \mathcal{B}\}$$

- critère *unweighted average linkage* :

$$D_{\text{UA}}(\mathcal{A}, \mathcal{B}) = \frac{1}{\#\mathcal{A} \#\mathcal{B}} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{A} \times \mathcal{B}} d(\mathbf{x}, \mathbf{y})$$

- critère de Ward :

$$D_{\text{W}}(\mathcal{A}, \mathcal{B}) = \frac{\#\mathcal{A} \#\mathcal{B}}{\#\mathcal{A} + \#\mathcal{B}} \|\mathbf{m}_{\mathcal{A}} - \mathbf{m}_{\mathcal{B}}\|^2$$

où  $\mathbf{m}_{\mathcal{A}} = \frac{1}{\#\mathcal{A}} \sum_{\mathbf{x} \in \mathcal{A}} \mathbf{x}$  et  $\mathbf{m}_{\mathcal{B}} = \frac{1}{\#\mathcal{B}} \sum_{\mathbf{x} \in \mathcal{B}} \mathbf{x}$  sont les barycentres des deux groupes, et  $\#\mathcal{A}$  et  $\#\mathcal{B}$  leur cardinal.

Notons que même si  $d$  vérifie les propriétés d'une distance sur un espace métrique, ce n'est pas forcément le cas de  $D$ .

Pour que deux groupes soient proches au sens du critère *single linkage*, il suffit que deux de leurs points soient proches, alors qu'au sens du critère *complete linkage* il faut que les points les plus éloignés entre les deux groupes soient proches (donc que tous les points soient proches). Le critère *average linkage* porte sur la moyennes des distances entre toutes les paires de points des groupes.

Pour que deux groupes soient proches au sens du critère de Ward, il faut que leurs barycentres soient proches, la notion de proximité étant pondérée par le nombre de points dans les groupes. Pour une même distance euclidienne entre barycentres, deux groupes de tailles similaires donneront un  $D$  plus élevé que deux groupes de tailles très différentes. En effet, dans le premier cas ( $\#\mathcal{A} \simeq \#\mathcal{B}$ ),  $\#\mathcal{A} \#\mathcal{B} / (\#\mathcal{A} + \#\mathcal{B}) \simeq \#\mathcal{A} / 2$ , et dans le second cas ( $\#\mathcal{A} \gg \#\mathcal{B}$ ),  $\#\mathcal{A} \#\mathcal{B} / (\#\mathcal{A} + \#\mathcal{B}) \simeq \#\mathcal{B}$ . Le critère de Ward tend donc à rapprocher les « gros » groupes des « petits » groupes.

Par ailleurs, on peut noter que :

$$D_{\text{W}}(\mathcal{A}, \mathcal{B}) = \sum_{\mathbf{x} \in \mathcal{A} \cup \mathcal{B}} \|\mathbf{x} - \mathbf{m}_{\mathcal{A} \cup \mathcal{B}}\|^2 - \sum_{\mathbf{x} \in \mathcal{A}} \|\mathbf{x} - \mathbf{m}_{\mathcal{A}}\|^2 - \sum_{\mathbf{x} \in \mathcal{B}} \|\mathbf{x} - \mathbf{m}_{\mathcal{B}}\|^2$$



(à faire en exercice, il suffit de développer les  $\| \cdot \|^2$  dans le membre de droite et d'utiliser l'égalité  $(\#\mathcal{A} + \#\mathcal{B})m_{\mathcal{A} \cup \mathcal{B}} = \#\mathcal{A}m_{\mathcal{A}} + \#\mathcal{B}m_{\mathcal{B}}$ , les deux groupes étant disjoints).

Cela signifie que  $D_W$  mesure l'écart entre la dispersion des deux groupes autour de leur moyenne commune (l'inertie inter-groupes) et la somme des dispersions de chaque groupe (les inerties intra-groupes).

### 3.1.2 Algorithme de construction du dendrogramme

L'ensemble des observations étant muni d'une mesure de dissimilarité  $d$  et les groupes d'une mesure  $D$ , on peut appliquer récursivement l'algorithme suivant :

1. initialisation : chaque observation définit un groupe ;
2. jusqu'à ce qu'il ne reste qu'un seul groupe, on fusionne successivement les deux groupes les plus proches selon la mesure de dissimilarité  $D$ .

Lorsque l'étape 2 est itérée, il faut calculer la distance entre le nouveau groupe obtenu par fusion et les groupes restants de l'itération précédente.

On construit ainsi un arbre binaire appelé dendrogramme :

1. les feuilles de l'arbre sont les observations, placées sur l'axe des abscisses,
2. pour chaque paire de groupes  $(\mathcal{A}, \mathcal{B})$  fusionnant, on trace des segments verticaux issus des deux groupes, jusqu'à la distance  $D(\mathcal{A}, \mathcal{B})$ , et on représente la fusion entre  $(\mathcal{A}$  et  $\mathcal{B})$  par un segment horizontal à la « hauteur »  $D(\mathcal{A}, \mathcal{B})$  figurant sur l'axe des ordonnées.

On construit donc une structure hiérarchique en partant des observations isolées, et à chaque étape, la hauteur des segments augmente. Ceci justifie l'appellation « méthode hiérarchique ascendante ».

Du point de vue du temps de calcul, l'algorithme de construction de la classification hiérarchique ascendante nécessite au moins de calculer les distances entre points deux à deux et de les trier, ce qui est relativement lent. Par ailleurs, l'occupation mémoire peut également être importante si on n'y prend pas garde, si par exemple on calcule à l'initialisation de l'algorithme le tableau des distances entre toutes les observations deux à deux.

### 3.1.3 Nombre de groupes

Nous n'avons pas encore répondu à la question du nombre de groupes à trouver. Si on fixe a priori le nombre de groupes à une valeur  $K$ , on peut arrêter l'algorithme précédent lorsqu'il ne reste plus que  $K$  groupes. De manière équivalente, cela revient à couper le dendrogramme à une certaine hauteur de manière à intercepter  $K$  segments verticaux, qui définissent alors les groupes. Le dendrogramme peut également fournir une indication sur le nombre de groupes : s'il présente un « saut » important entre deux valeurs  $a$  et  $b$ , cela signifie que la dissimilarité entre observations dans chaque groupe est bien moindre que la dissimilarité entre observations d'au moins deux groupes fusionnés. On a donc intérêt à intercepter le dendrogramme à une hauteur comprise entre  $a$  et  $b$ , et on en déduit le nombre de groupes identifiés.

### 3.1.4 Illustration

La figure 3.1 montre des classifications hiérarchiques obtenues à partir d'un ensemble de points très simple, de manière à illustrer la construction du dendrogramme. Le nombre de groupes à trouver a été fixé à deux. Il s'agit donc des deux dernières groupes fusionnant dans la construction du dendrogramme. La mesure de dissimilarité  $d$  est la distance euclidienne. On voit que le critère *single linkage* ne convient pas ici. Il souffre du problème de « chaînage » : il a tendance à progressivement fusionner les groupes liés par une chaîne de points. Le critère *complete linkage* est basé sur la plus grande distance entre points de deux groupes, ce qui explique que les points 4 et 5 appartiennent à deux groupes différents. Les critères *average linkage* et de *Ward*, basés sur des moyennes, fournissent des groupes homogènes et relativement bien séparés les uns des autres. Notez l'aspect similaire des dendrogrammes construits selon ces deux critères.

D'après le paragraphe 3.1.3, on pourrait fixer le nombre de groupes à sept dans la classification *single linkage* (groupe {3, 4, 5} et les six autres observations dans autant de groupes), à trois, peut-être deux dans la classification *complete linkage* (groupes {1, 2, 3, 4}, {5, 7}, {6, 8, 9}), et à deux dans les classifications *average linkage* et *Ward* (groupes {1, 2, 3, 4, 5} et {6, 7, 8, 9}). On se rend compte que selon la distribution des observations, l'un ou l'autre des critères devra être préféré. Toute la difficulté pratique est de choisir lequel.

La figure 3.2 est une autre illustration du phénomène de chaînage.

## 3.2 Partitionnement en $K$ -moyennes

Il s'agit d'un algorithme très utilisé en pratique dans de nombreuses applications scientifiques et industrielles car il est simple à mettre en œuvre et est relativement rapide.

Dans l'expression « partitionnement en  $K$ -moyennes » (*K-means*),  $K$  désigne le nombre de groupes à identifier. Il s'agit d'un paramètre fixé a priori par l'utilisateur. L'objectif est de trouver une partition  $(\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_K)$  de l'ensemble des observations  $(\mathbf{x}_n)_{1 \leq n \leq N}$  qui minimise la fonction suivante, souvent appelée « inertie » :

$$E(\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_K) = \sum_{j=1}^K \sum_{\mathbf{x} \in \mathcal{C}_j} \|\mathbf{x} - \mathbf{m}_j\|_2^2$$

où  $\mathbf{m}_j = \frac{1}{\#\mathcal{C}_j} \sum_{\mathbf{x} \in \mathcal{C}_j} \mathbf{x}$  est la moyenne des observations appartenant au groupe  $\mathcal{C}_j$  ( $j$  entre 1 et  $K$ ), d'où le nom «  $K$ -moyennes ». Rappelons que  $\|\cdot\|_2$  désigne la norme euclidienne.

La fonction  $E$  est la somme des dispersions des groupes, chaque dispersion étant mesurée par la somme des carrés des distances des points du groupe à son barycentre, c'est-à-dire une inertie. On cherche à classer les observations en  $K$  groupes minimisant la somme des inerties des groupes.

### 3.2.1 Algorithme de Lloyd

L'algorithme de Lloyd (1957) permet de trouver une partition correspondant à un minimum local de l'inertie  $E$ , en un sens précisé dans la démonstration ci-dessous. Il est illustré par la figure 3.3.

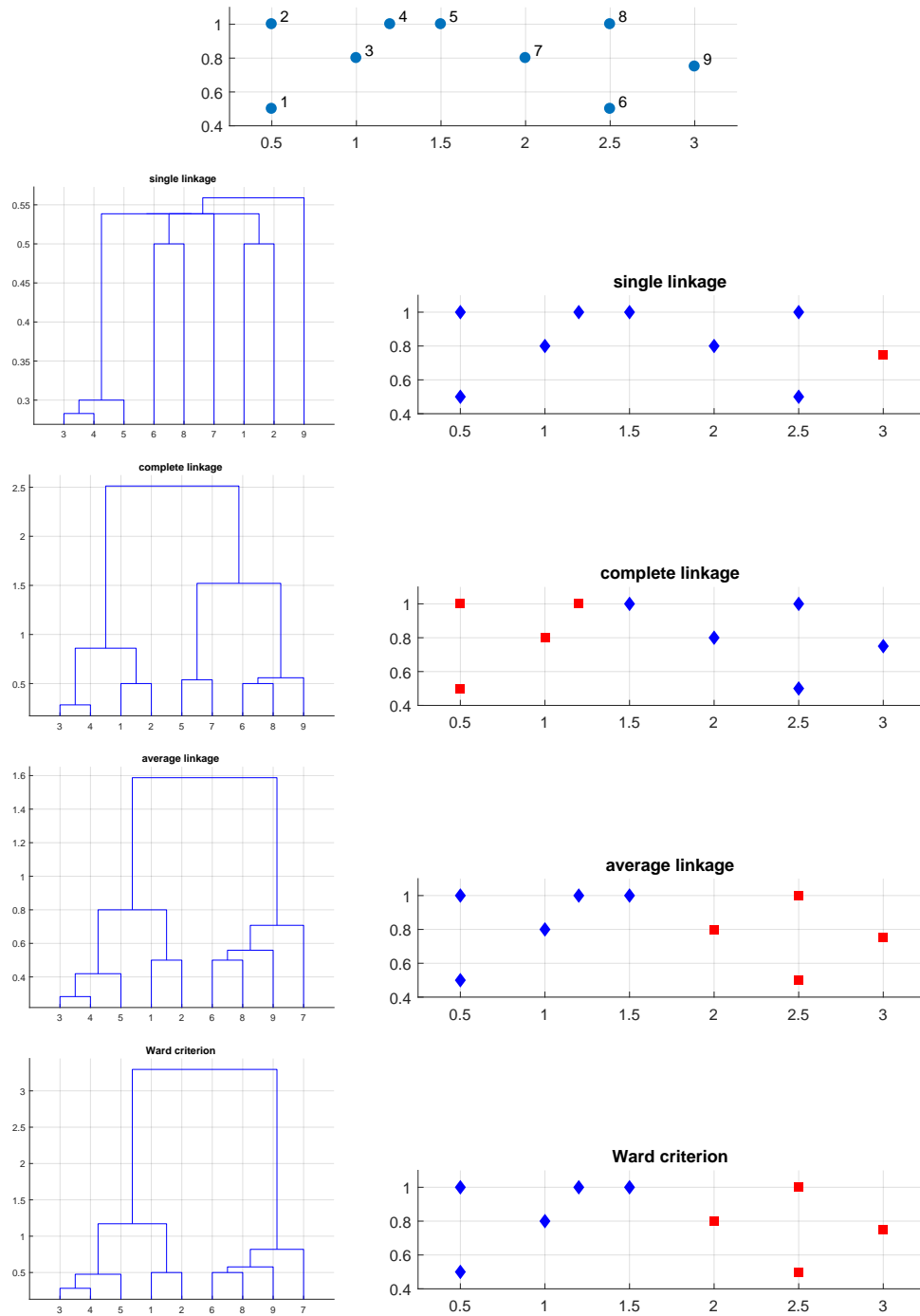


FIGURE 3.1 – Exemples de partitionnement avec quatre mesures de dissimilarité entre groupes, sur le même ensemble de point. De haut en bas : ensemble d'observations à partitionner, critère single linkage, critère complete linkage, critère average linkage, critère de Ward. Le dendrogramme est à gauche et la partition en deux groupes obtenue à droite.

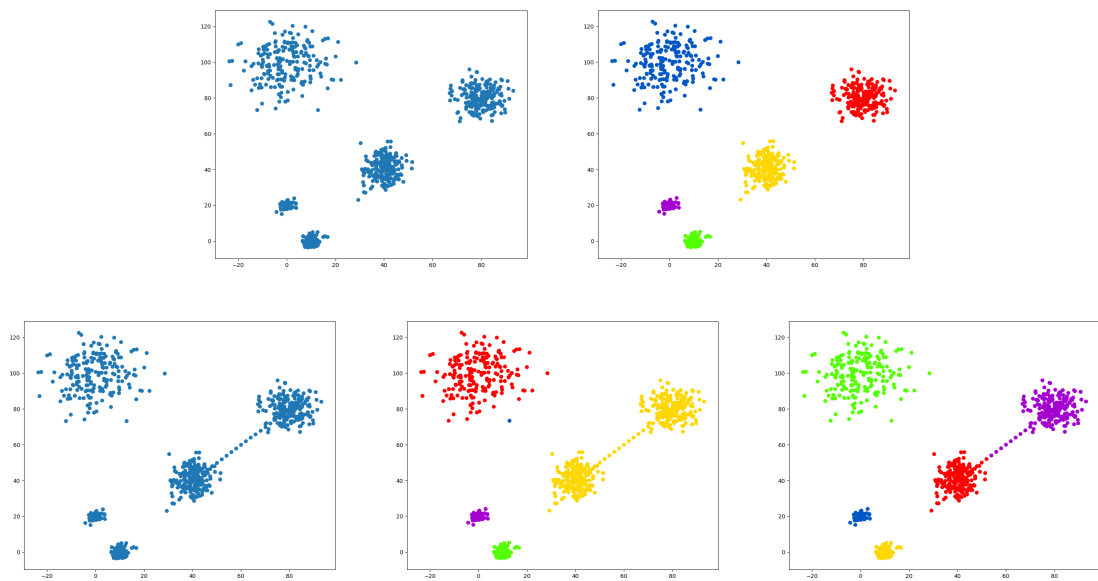


FIGURE 3.2 – *Illustration du phénomène de chaînage. Sur la première ligne, le partitionnement de l'ensemble d'observations (à gauche) en cinq groupes avec le critère single linkage (à droite) semble cohérent. Sur la deuxième ligne, on voit que la présence des points « reliant » deux des groupes (à gauche) entraîne leur fusion précoce par single linkage (au milieu) : un des groupes est formé d'un point isolé (près du groupe en haut à gauche). La classification selon le critère de Ward (à droite) reste cohérente, car celui-ci est basé sur l'inertie des groupes identifiés, et tend à favoriser la compacité des groupes identifiés. Figure à voir en couleur.*

Voici sa description.

Initialisation : choix aléatoire de  $K$  « moyennes »  $\mathbf{m}_j^0$

Itérer les trois étapes suivantes, pour  $t \geq 0$  :

1. pour tout  $1 \leq j \leq K$ , définir  $\mathcal{C}_j^t$  comme l'ensemble des observations  $\mathbf{x}$  plus proche de  $\mathbf{m}_j^t$  que des autres moyennes,
2. étant donnée une partition  $(\mathcal{C}_1^t, \mathcal{C}_2^t, \dots, \mathcal{C}_K^t)$ , calculer les  $K$  moyennes  $(\mathbf{m}_j^{t+1})_{1 \leq k \leq K}$  de chaque groupe,
3. incrémenter  $t$ .

Nous allons démontrer la proposition suivante.

**Proposition 3.1** *L'algorithme de Lloyd converge vers un minimum local de l'inertie  $E$  en un nombre fini d'itérations.*

La démonstration repose sur l'évolution de l'inertie entre deux itérations. Auparavant, on démontre une propriété classique de la norme euclidienne. Rappelons qu'on désigne par  $\#A$  le cardinal d'un ensemble fini  $A$ .

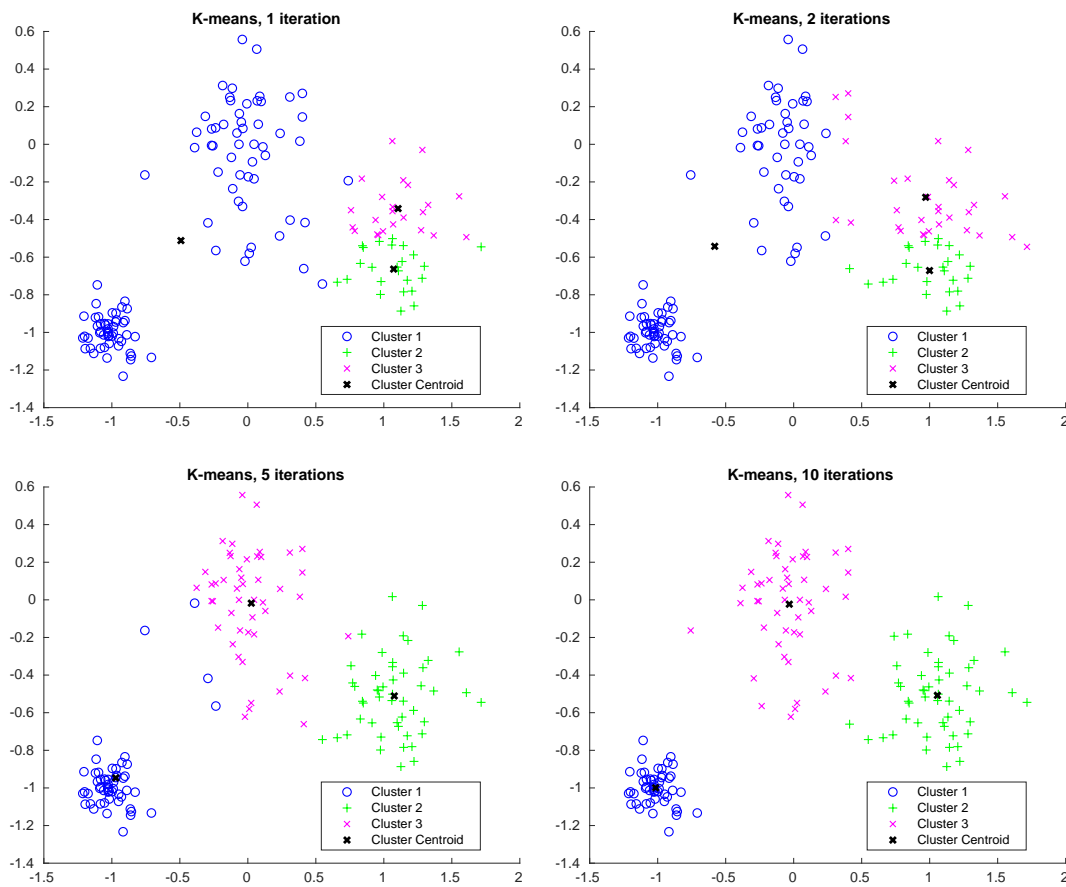


FIGURE 3.3 – Algorithme de Lloyd : illustration. Évolution des moyennes (barycentres des groupes) et affectation des observations au cours des itérations. Attention, sur chaque graphique sont représentés les clusters à l'étape  $t - 1$  et les barycentres à l'étape  $t$ , conformément au déroulement de l'algorithme.

**Lemme 1** Soit  $A$  un ensemble fini de points dans un espace euclidien. La fonction

$$y \mapsto \sum_{x \in A} \|x - y\|_2^2$$

définie sur tout l'espace est minimale pour la moyenne des points de  $A$  :

$$y = \frac{1}{\#A} \sum_{x \in A} x$$

*Démonstration*

On développe successivement, pour tout  $y$  :

$$\begin{aligned}
\sum_{\mathbf{x} \in A} \|\mathbf{x} - \mathbf{y}\|_2^2 &= \sum_{\mathbf{x} \in A} (\|\mathbf{x}\|_2^2 + \|\mathbf{y}\|_2^2 - 2\mathbf{x} \cdot \mathbf{y}) \\
&= \sum_{\mathbf{x} \in A} \|\mathbf{x}\|_2^2 + \#A \|\mathbf{y}\|_2^2 - 2 \left( \sum_{\mathbf{x} \in A} \mathbf{x} \right) \cdot \mathbf{y} \\
&= \sum_{\mathbf{x} \in A} \|\mathbf{x}\|_2^2 + \frac{1}{\#A} \left\| \#A \mathbf{y} - \sum_{\mathbf{x} \in A} \mathbf{x} \right\|_2^2 - \frac{1}{\#A} \left\| \sum_{\mathbf{x} \in A} \mathbf{x} \right\|_2^2
\end{aligned}$$

est bien minimal pour  $\mathbf{y} = \frac{1}{\#A} \sum_{\mathbf{x} \in A} \mathbf{x}$  qui annule le seul terme dépendant de  $\mathbf{y}$ .  $\square$

On peut à présent démontrer la proposition 3.1.

### Démonstration

Avec les notations de la description de l'algorithme, en notant  $E^t$  l'inertie à la fin de l'itération  $t$ , on peut écrire :

$$\begin{aligned}
E^t &= \sum_{j=1}^K \sum_{\mathbf{x} \in \mathcal{C}_j^t} \|\mathbf{x} - \mathbf{m}_j^{t+1}\|_2^2 \\
&\geq \sum_{j=1}^K \sum_{\mathbf{x} \in \mathcal{C}_j^{t+1}} \|\mathbf{x} - \mathbf{m}_j^{t+1}\|_2^2
\end{aligned}$$

car la partition  $(\mathcal{C}_1^{t+1}, \mathcal{C}_2^{t+1}, \dots, \mathcal{C}_K^{t+1})$  est justement formée, à l'étape  $t+1$ , de manière à ce que chaque  $\mathbf{x}$  soit associé au  $\mathbf{m}_j^{t+1}$  le plus proche.

Comme  $\mathbf{m}_j^{t+2}$  minimise la fonction qui à  $\mathbf{y}$  associe  $\sum_{\mathbf{x} \in \mathcal{C}_j^{t+1}} \|\mathbf{x} - \mathbf{y}\|_2^2$  (cf lemme 1), on déduit l'inégalité :

$$E^t \geq \sum_{j=1}^K \sum_{\mathbf{x} \in \mathcal{C}_j^{t+1}} \|\mathbf{x} - \mathbf{m}_j^{t+2}\|_2^2$$

d'où  $E^t \geq E^{t+1}$ .

La suite des inerties  $E^t$  est donc décroissante. Comme il n'y a qu'un nombre fini de possibilités pour former  $K$  groupes parmi  $N$  observations (il s'agit du nombre de Stirling de seconde espèce  $s(N, K)$ ), il existe un  $t^*$  tel que :  $E^{t^*} = E^{t^*+1}$ . Pour obtenir cette égalité, les deux inégalités précédemment établies doivent être des égalités, donc pour tout  $j$ ,  $\mathbf{m}_{t^*+1}^j = \mathbf{m}_{t^*+2}^j$ , et pour tout  $j$ ,  $\mathcal{C}_j^{t^*+1} = \mathcal{C}_j^{t^*}$ . Les assignations des observations aux classes ne changent donc plus pour  $t > t^*$ . C'est en ce sens qu'on parle de minimum local de l'inertie (et pas dans un sens habituel faisant intervenir le voisinage dans un espace métrique : ici l'inertie est définie sur l'ensemble fini des partitions en  $K$  groupes). Une autre initialisation pourrait conduire à une partition (et une valeur de  $E^{t^*}$ ) différente.  $\square$

### 3.2.2 Discussion

La description de l'algorithme de Lloyd ne repose pas explicitement sur la notion de norme euclidienne : cet algorithme ne fait appel qu'à une distance et à la notion de moyenne. Pourtant, la convergence de l'algorithme est bien assurée par le lemme 1, qui caractérise la moyenne à l'aide de la norme euclidienne. Utiliser une norme non-euclidienne n'assure plus la convergence : il faut alors employer d'autres algorithmes que celui de Lloyd.

L'algorithme de Lloyd converge vers un minimum local de l'inertie : l'initialisation étant aléatoire, un autre minimum local peut être trouvé à chaque exécution. Il est d'usage d'exécuter l'algorithme plusieurs fois, et de garder la partition d'inertie minimale parmi celles trouvées.

Concernant le nombre d'itérations nécessaire avant de converger, la démonstration nous dit qu'il est borné par le nombre de Stirling de seconde espèce  $s(N, K)$ , dont on dispose de l'équivalent :

$$s(N, K) \sim_{N \rightarrow +\infty} K^N / K!$$

à  $K$  fixé. En pratique, le nombre de groupes  $K$  est bien inférieur au nombre  $N$  d'observations à classer. Des grandeurs typiques, qui peuvent être bien plus grandes dans certaines applications, sont par exemple  $N = 1000$ ,  $K = 10$ , d'où un nombre astronomique de possibilités à tester (rappelons que le nombre d'atomes dans l'univers est estimé à  $10^{80}$ ). Il est illusoire de tester exhaustivement toutes ces possibilités : un algorithme guidant la recherche du minimum de l'inertie est nécessaire. En pratique l'algorithme de Lloyd s'arrête bien avant que le nombre d'itérations atteigne cette borne ; il permet donc de guider efficacement la recherche. De plus, il est possible d'arrêter l'algorithme avant convergence au bout de quelques itérations car en général seuls quelques points mal classés subsistent alors.

Comme l'inertie mesure la dispersion de chaque groupe autour de sa moyenne par l'intermédiaire de la norme euclidienne, l'algorithme des  $K$ -moyennes sera adapté à l'identification de groupes de forme sphérique, bien représentés par leur moyenne.

Le paramètre  $K$  est critique, et bien entendu on ne sait pas à l'avance combien de groupes sont recherchés. On se rend compte qu'une partition optimale à  $K + 1$  groupes a une inertie inférieure à celle d'une partition optimale à  $K$  groupes. En effet, considérons la partition optimale à  $K$  groupes et formons une partition à  $K + 1$  groupes en retirant une observation  $\tilde{x}$  du  $K$ -ème groupe pour en faire le  $K + 1$ -ème groupe. L'inertie de la partition optimale à  $K$  groupes s'écrit :

$$\sum_{j=1}^K \sum_{\mathbf{x} \in \mathcal{C}_j} \|\mathbf{x} - \mathbf{m}_j\|_2^2 = \sum_{j=1}^{K-1} \sum_{\mathbf{x} \in \mathcal{C}_j} \|\mathbf{x} - \mathbf{m}_j\|_2^2 + \sum_{\mathbf{x} \in \mathcal{C}_K \setminus \{\tilde{x}\}} \|\mathbf{x} - \mathbf{m}_K\|_2^2 + \|\tilde{x} - \mathbf{m}_K\|_2^2$$

où les  $\mathbf{m}_j$  sont les moyennes des  $\mathcal{C}_j$ . Les barycentres des groupes  $\mathcal{C}_1, \dots, \mathcal{C}_{K-1}$  ne changent pas, et si on note  $\mathbf{m}'_K$  le barycentre du groupe  $\mathcal{C}_K \setminus \{\tilde{x}\}$  et  $\mathbf{m}'_{K+1}$  celui de  $\mathcal{C}_{K+1}$  ( $\mathbf{m}'_{K+1} = \tilde{x}$  tout simplement), alors :

- $\sum_{\mathbf{x} \in \mathcal{C}_K \setminus \{\tilde{x}\}} \|\mathbf{x} - \mathbf{m}_K\|_2^2 \geq \sum_{\mathbf{x} \in \mathcal{C}_K \setminus \{\tilde{x}\}} \|\mathbf{x} - \mathbf{m}'_K\|_2^2$  d'après le lemme 1,
- $\|\tilde{x} - \mathbf{m}_K\|_2^2 \geq 0 = \|\tilde{x} - \mathbf{m}'_{K+1}\|_2^2$ .

L'inertie minimale à  $K$  groupes est donc supérieure à une inertie obtenue pour  $K + 1$  groupes. Elle est donc également supérieure à l'inertie minimale parmi toutes les partitions à  $K + 1$  groupes.

Par conséquent, si on trace le graphe de l'inertie minimale en fonction de  $K$ , on observe une courbe décroissante. En pratique, la partition optimale trouvée par l'algorithme de Lloyd n'est pas l'optimum global de l'inertie, donc la courbe peut ne pas être strictement décroissante. On peut envisager de choisir comme nombre de groupes la valeur de  $K$  à partir de laquelle l'inertie ne décroît plus franchement. Il s'agit de la méthode du coude (la courbe marque un coude), ou *elbow method*, que l'on reverra lors des travaux pratiques.

### 3.3 Méthodes de partitionnement basées sur la densité

On peut voir les observations à partitionner comme des réalisations de variables aléatoires. Le problème du partitionnement est alors lié à celui d'estimer (implicitement) la densité de probabilité sous-jacente : un groupe sera identifié comme un ensemble d'observations dans une région de densité élevée.

#### 3.3.1 DBSCAN

DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*) est un algorithme de partitionnement très utilisé datant de 1996.

Il peut être décrit de la manière suivante, avec les paramètres  $\varepsilon > 0$  et  $N$  entier :

1. Pour chaque observation, chercher les observations (les « voisins ») à une distance inférieure à  $\varepsilon$ , et définir les observations possédant plus de  $N$  voisins comme les « points centraux »;
2. Construire le graphe de voisinage des points centraux, et définir les groupes comme les composantes connexes de ce graphe;
3. Assigner les observations qui ne sont pas des points centraux au groupe du point central le plus proche parmi les observations voisines, et les assigner au bruit si aucun voisin n'est point central.

Rappelons que le graphe de voisinage des points centraux est le graphe dont les sommets sont les points centraux, pour lequel deux sommets sont liés par une arête s'ils sont voisins (c'est-à-dire à une distance inférieure à  $\varepsilon$ ). Deux sommets du graphe sont dans la même composante connexe s'il existe une suite d'arêtes les connectant.

Dans l'étape 2, on ne considère que les points centraux : deux points centraux sont liés s'ils sont voisins l'un de l'autre. Le graphe résultant présente des composantes connexes : des ensembles de points centraux qui ne sont liés par aucun chemin à une autre composante connexe.

Dans l'étape 1, on voit que les points centraux sont les observations se situant dans des zones de forte densité.



Pour assurer une complexité optimale en occupation mémoire et en temps de calcul, il faut implanter l'algorithme de manière réfléchie : trouver les observations voisines et les composantes connexes du graphe de voisinage de manière efficace nécessite un peu de travail.

L'algorithme présente deux paramètres :  $\varepsilon$  et  $N$ . Il est déterministe mais le résultat est susceptible de dépendre de l'ordre de parcours des observations.

Les avantages de l'algorithme DBSCAN par rapport aux  $k$ -moyennes sont qu'il ne nécessite pas de fixer le nombre de groupes *a priori*, il peut trouver des groupes de forme arbitraire, et il peut « laisser de côté » des points aberrants (le bruit).

Les inconvénients de DBSCAN sont notamment sa dépendance au choix de la mesure de la distance entre les observations (comme tous les algorithmes vus dans ce chapitre), et sa difficulté à identifier des groupes de densités différentes car les paramètres  $\varepsilon$  et  $N$  sont uniformes sur toute la base d'observations.

La figure 3.4 illustre le comportement de DBSCAN.

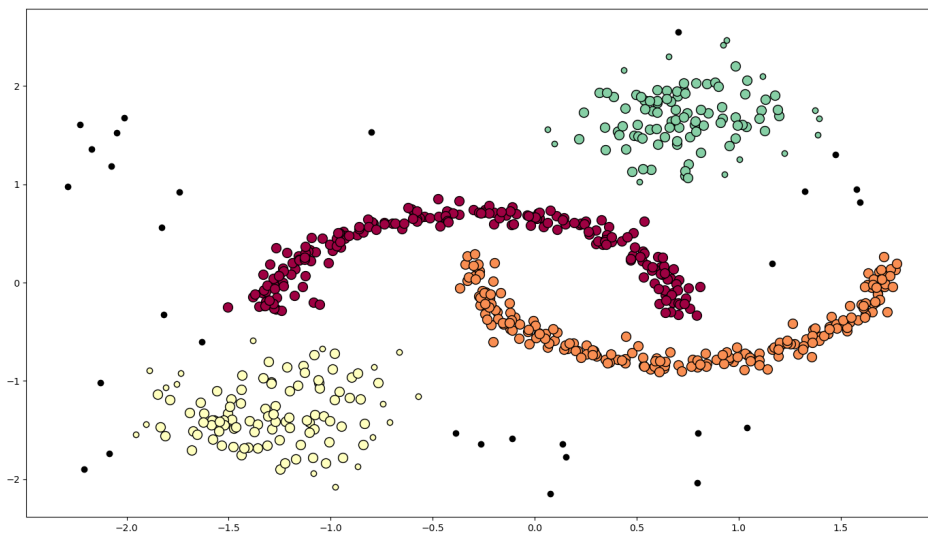


FIGURE 3.4 – Un exemple de partitionnement obtenu par DBSCAN (implantation de scikit-learn,  $\varepsilon = 0.3$ ,  $N = 10$ ). Quatre groupes ont été identifiés, les observations marquées par un grand cercle de couleur sont les points centraux (étape 1 de l'algorithme), celles marquées par un petit cercle de couleur ne sont pas des points centraux mais ont été affectés à une classe (étape 3). Les observations en noir sont considérées comme du bruit par l'algorithme. Figure à voir en couleur.

### 3.3.2 Mean shift

L'algorithme *Mean shift* (décalage de la moyenne) cherche à trouver des modes (des extrema) dans la distribution des observations. Il est très utilisé dans de nombreuses applications dans le domaine de la vision par ordinateur.

On verra au chapitre 5 (section 5.1.2) que l'on peut évaluer la densité de probabilité sous-jacente à un ensemble d'observations  $(\mathbf{x}_n)_{1 \leq n \leq N}$  par la méthode des fenêtres de Parzen. On suppose disposer d'une fenêtre, c'est-à-dire une fonction  $W$  concentrée autour de 0, d'intégrale 1 sur l'espace des observations.

L'estimation de la densité en un point  $\mathbf{x}$  de l'espace des observations est donnée par :

$$p_W(\mathbf{x}) = \frac{1}{Nh^d} \sum_{n=1}^N W\left(\frac{\mathbf{x} - \mathbf{x}_n}{h}\right)$$

où  $h$  est appelée « largeur de bande » (*bandwidth*), et où la somme est normalisée de manière à ce que  $p_W$  s'intègre à 1 sur tout l'espace.

On suppose que la fenêtre est gaussienne :

$$W(\mathbf{x}) = W_0 \exp(-\|\mathbf{x}\|_2^2/2)$$

avec  $W_0$  fixé de manière à ce que  $\int W = 1$ . Dans ce cas, le gradient de  $W$  vérifie la relation :  $\nabla W(\mathbf{x}) = -\mathbf{x}W(\mathbf{x})$ .

Nous allons identifier les extrema locaux de cette densité par une méthode de « montée de gradient » (voir annexe B.3).

On calcule :

$$\begin{aligned} \nabla p_W(\mathbf{x}) &= -\frac{1}{Nh^d} \sum_{n=1}^N \frac{\mathbf{x} - \mathbf{x}_n}{h^2} W\left(\frac{\mathbf{x} - \mathbf{x}_n}{h}\right) \\ &= \frac{1}{Nh^d} \sum_{n=1}^N W\left(\frac{\mathbf{x}_n - \mathbf{x}}{h}\right) \frac{\mathbf{x}_n}{h^2} - p_W(\mathbf{x}) \frac{\mathbf{x}}{h^2} \end{aligned}$$

D'où

$$\frac{h^2 \nabla p_W(\mathbf{x})}{p_W(\mathbf{x})} = \frac{\sum_{n=1}^N W\left(\frac{\mathbf{x}_n - \mathbf{x}}{h}\right) \mathbf{x}_n}{\sum_{n=1}^N W\left(\frac{\mathbf{x}_n - \mathbf{x}}{h}\right)} - \mathbf{x}$$

Une étape de montée de gradient à pas variable consisterait à remplacer  $\mathbf{x}$  par  $\mathbf{x} + \lambda(\mathbf{x}) \nabla p_W(\mathbf{x})$  où  $\lambda(\mathbf{x}) > 0$  est le pas de la montée. Si la montée de gradient converge bien, les itérés successifs se rapprochent de plus en plus d'un maximum local.

Si on note :

$$m_h(\mathbf{x}) = \frac{\sum_{n=1}^N W\left(\frac{\mathbf{x}_n - \mathbf{x}}{h}\right) \mathbf{x}_n}{\sum_{n=1}^N W\left(\frac{\mathbf{x}_n - \mathbf{x}}{h}\right)} = \mathbf{x} + \frac{h^2 \nabla p_W(\mathbf{x})}{p_W(\mathbf{x})}$$

remplacer  $\mathbf{x}$  par  $m_h(\mathbf{x})$  revient à faire une étape de montée de gradient avec  $\lambda(\mathbf{x}) = \frac{h^2}{p_W(\mathbf{x})}$ . La fonction  $\lambda$  assure un pas de montée grand lorsque  $p_W$  est petit (on est loin du maximum, on peut accélérer) et petit lorsque  $p_W$  est grand (on s'approche du maximum, il faut ralentir). La quantité  $m_h(\mathbf{x})$  est la moyenne des observations pondérée par leur distance à  $\mathbf{x}$ .

Ce développement mathématique nous suggère qu'en itérant ce processus, la suite obtenue converge vers un maximum local de la densité des observations  $p_W$ . On peut affecter à un groupe les observations qui appartiennent au bassin d'attraction du même maximum local.

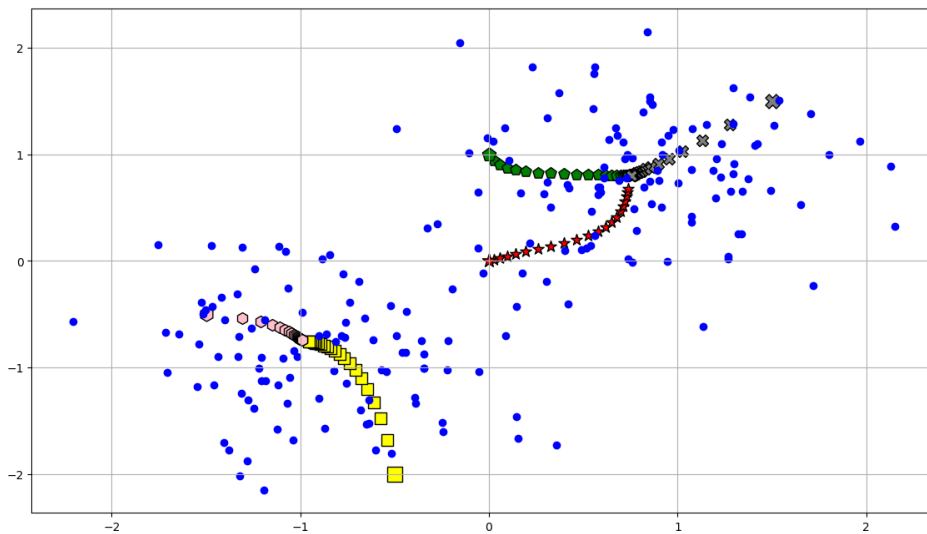


FIGURE 3.5 – Cinq exemples d'évolution des itérés de l'algorithme *mean shift* (fenêtre gaussienne, largeur de bande  $h = 0,5$ ). Les cinq observations initiales sont identifiées par un symbole plus grand : on voit qu'au fur et à mesure, la suite des moyennes converge vers un des deux extrema locaux de la densité des observations. Le partitionnement par *mean shift* consiste à associer au même groupe les observations qui convergent vers le même point.

L'algorithme de partitionnement *mean shift* est décrit ci-dessous.

— pour chaque observation  $\mathbf{x}_n$ , itérer :

$$\begin{cases} \mathbf{x}_n^0 = \mathbf{x}_n \\ \mathbf{x}_n^{i+1} = m_h(\mathbf{x}_n^i) \text{ pour tout } i > 0 \end{cases}$$

jusque convergence ;

— affecter à un même groupe les observations qui ont convergé vers la même limite

La figure 3.5 montre, dans un cas pratique, les itérés successifs obtenus à partir de quelques points d'initialisation. On peut rendre la discussion précédente un peu plus rigoureuse et démontrer que *mean shift* converge effectivement. Nous avons supposé utiliser une fenêtre gaussienne. En fait, on voit que si  $\nabla W(\mathbf{x}) = -c\mathbf{x}K(\mathbf{x})$  (où  $c$  est une constante positive), alors l'algorithme *mean shift* utilisant la fenêtre  $K$  peut toujours être interprété comme une montée de gradient sur la densité estimée par la méthode de Parzen avec la fenêtre  $W$ . N'importe quelle fonction ne convient pas : il faut que  $W$  et  $K$  soient des fenêtres (c'est-à-dire concentrées autour de 0). Ceci justifie l'utilisation fréquente de la fenêtre

$$K(\mathbf{x}) = \begin{cases} 1 & \text{si } |\mathbf{x}| \leq 1 \\ 0 & \text{sinon} \end{cases}$$

dans l'algorithme *mean shift*, au lieu d'une fenêtre gaussienne. La convergence est assurée car la fenêtre  $W$  associée est la fonction d'Epanechnikov :

$$W(\mathbf{x}) = \begin{cases} \frac{3}{4}(1 - x^2) & \text{si } |x| \leq 1 \\ 0 & \text{sinon} \end{cases}$$

L'algorithme *mean shift* peut trouver des groupes de forme quelconque. En pratique, le choix de la largeur de bande  $h$  peut être critique. Néanmoins, la largeur de bande a une interprétation « physique » : elle dépend de la densité des observations. Des heuristiques existent pour choisir sa valeur *a priori*, contrairement au nombre de classes  $K$  qui est le paramètre de l'algorithme des  $K$ -moyennes. Enfin, le calcul de  $m_h$  est lourd, ce qui rend l'algorithme assez lent en pratique.

La figure 3.6 montre un exemple de partitionnement par *mean shift*.

### 3.4 Pour approfondir...

**Distance d'édition** La distance d'édition (ou de Levenshtein) évoquée en section 3.1.1 fait l'objet d'un exercice dans le polycopié disponible à l'URL suivante (p. 13 et 76) :

<https://members.loria.fr/FSur/enseignement/RO/>

**Complexité algorithmique de l'algorithme des  $K$ -moyennes** Pour certaines distributions « pathologiques » des observations, le nombre d'itérations nécessaires pour que l'algorithme de Lloyd converge est de l'ordre de grandeur de  $2^N$ , comme l'indique la borne donnée par les nombres de Stirling. On peut alors se demander ce qui explique le succès de l'algorithme sur des données réelles, comme mentionné dans le paragraphe 3.2.2. L'article suivant nous donne une piste de compréhension :

D. Arthur, B. Manthey, H. Röglin, *Smoothed analysis of the  $k$ -means method*, Journal of the ACM, vol. 58, no 5, 2011.

Citons la conclusion de son résumé : “*if an arbitrary input data set is randomly perturbed, then the  $k$ -means method will run in expected polynomial time on that input set*”. Lorsqu'on étudie la complexité des algorithmes, une borne polynomiale est bien meilleure qu'une borne exponentielle qui bien souvent traduit l'impossibilité pratique d'utiliser l'algorithme. La propriété mentionnée signifie qu'il faut vraiment ne pas avoir de chance pour tomber sur un ensemble d'observations nécessitant un nombre exponentiel d'itérations. En effet, les perturbations aléatoires de cet ensemble « pathologique » nécessitent en moyenne un nombre polynomial d'itérations, et les observations réelles peuvent justement être vues comme réalisation d'un processus aléatoire à cause du bruit de mesure.

**Accélération des  $K$ -moyennes en pratique** La tâche chronophage dans l'algorithme de Lloyd est l'assignation de toutes les observations à la moyenne la plus proche (qu'il faut trouver). L'algorithme MiniBatch  $K$ -means (dont on utilisera l'implantation fournie dans la bibliothèque `scikit-learn`) extrait à chaque itération un échantillon aléatoire de l'ensemble des observations, et se sert uniquement de cet échantillon pour mettre à jour les moyennes. Une

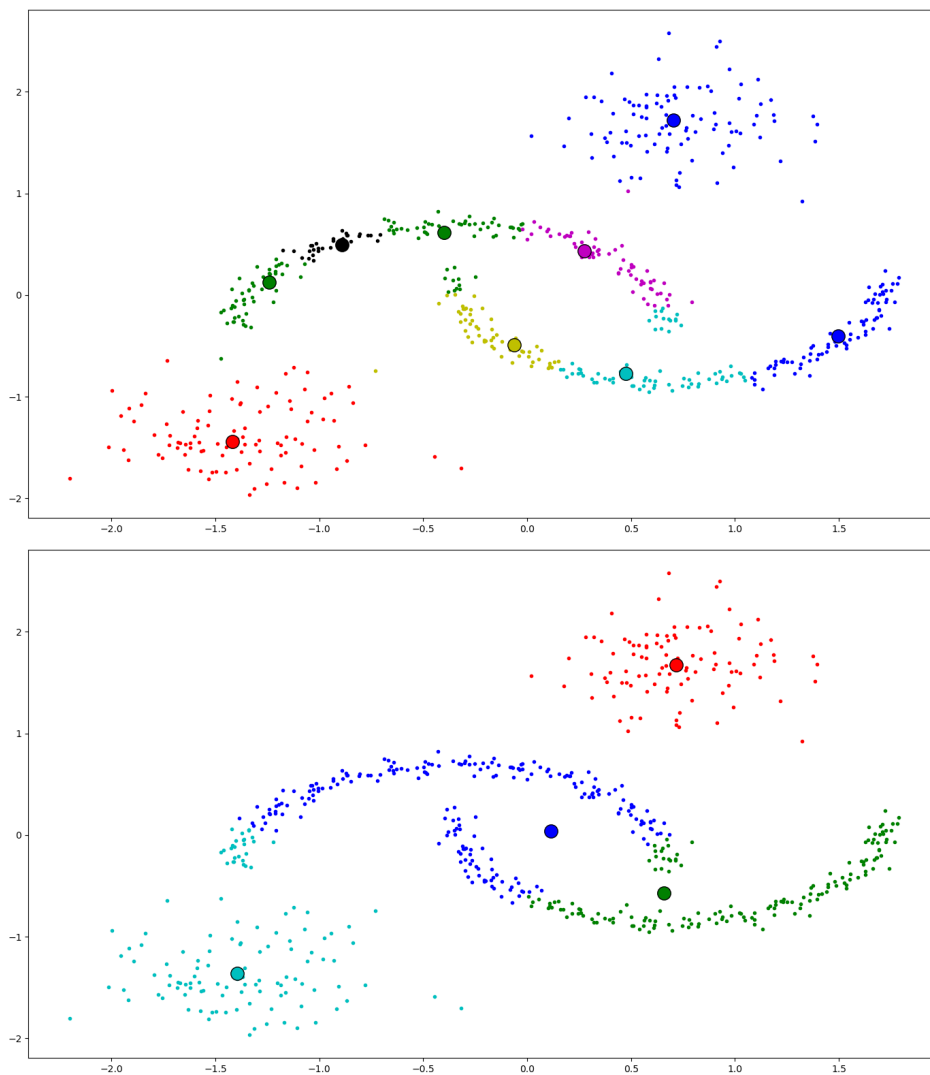


FIGURE 3.6 – Deux exemples de partitionnement obtenus par mean shift (implantation de scikit-learn). Première ligne : largeur de bande de 0,5; 9 groupes sont identifiés. Seconde ligne : 0,8; 4 groupes sont identifiés. On voit l'influence de la largeur de bande : plus elle est grande plus l'estimation de la densité est « lissée » et moins cette densité présente de maxima (voir la figure 5.2 au chapitre 5). Le disque de couleur montre la limite des itérés successifs des observations associées à chaque groupe : il s'agit d'un maximum de l'estimation de la densité de probabilité par la méthode des fenêtres de Parzen.

partition satisfaisante est alors trouvée bien plus rapidement que par l'algorithme de Lloyd, au prix d'une inertie un peu plus grande que la solution de Lloyd localement optimale. Mini-Batch  $K$ -means est décrit dans cet article :

D. Sculley, *Web-Scale K-Means Clustering*, proceedings of the 19th International Conference on World Wide Web, p. 1177-1178, 2010

téléchargeable depuis la documentation de scikit-learn :

<https://scikit-learn.org/stable/modules/clustering.html#mini-batch-kmeans>

**Lecture** Pour plus de détails sur les  $K$ -moyennes et le partitionnement en général, vous pouvez consulter :

Anil K. Jain, *Data clustering : 50 years beyond K-means*, Pattern Recognition Letters, vol. 31, no. 8, p. 651-666, 2010.

# Théorie statistique de la décision

Nous nous plaçons dans le cadre de l'apprentissage supervisé. On suppose disposer d'une base de  $N$  observations étiquetées  $(\mathbf{x}_n, y_n)_{1 \leq n \leq N}$ , avec  $y_n$  une étiquette qui prend une valeur scalaire ou vectorielle dans le cas de la régression, ou une valeur dans un ensemble fini de classes dans le cas de la classification. Rappelons qu'on cherche une fonction  $f_{\mathbf{w}}$  capable de prédire l'étiquette d'une nouvelle observation  $\mathbf{x}$  en se basant sur les observations étiquetées. Cette fonction dépend généralement d'un ensemble de paramètres  $\mathbf{w}$ .

La question à laquelle on essaie d'apporter des éléments de réponse est la suivante : comment choisir les paramètres  $\mathbf{w}$  de  $f_{\mathbf{w}}$  de manière à minimiser le risque d'erreur face à une nouvelle observation absente de la base initiale des  $N$  observations ? Nous savons depuis le chapitre 2 qu'il ne suffit pas de chercher  $\mathbf{w}$  tel que  $f_{\mathbf{w}}$  minimise le risque d'erreur sur la base d'apprentissage (le risque empirique), comme le montre l'exemple de l'apprentissage « par cœur ».

Dans ce chapitre, nous définissons le prédicteur optimal dans le cadre d'une modélisation probabiliste de l'ensemble des observations.

## 4.1 Minimisation du risque de prédiction

Comme dans le chapitre 2 (nous utilisons les notations de la section 2.2.1), nous introduisons la fonction de perte (*loss function*)  $\ell(y, h(\mathbf{x}))$  quantifiant le coût d'une erreur de prédiction sur l'observation  $\mathbf{x}$ , et nous cherchons le prédicteur  $h$  qui minimise le coût d'erreur moyen sur l'espace des observations possibles.

Ce coût moyen est défini comme l'espérance du coût d'erreur sous la loi de probabilité du couple  $(\mathbf{X}, Y)$  définie sur l'espace des observations ; il s'agit du risque moyen de prédiction  $R_p$  défini précédemment comme :

$$R_p(h) = E_{(\mathbf{X}, Y)}(\ell(Y, h(\mathbf{X})))$$

Supposons un instant que la loi de  $(\mathbf{X}, Y)$  est connue, et notons-la  $p(\mathbf{x}, y)$ . Bien entendu, toute la difficulté est qu'elle est, en pratique, inconnue. Que peut-on dire d'un classifieur  $h$

qui minimiserait le risque moyen  $R_p$  ?

Nous discutons séparément régression et classification supervisée, les fonctions de coût  $\ell$  étant différentes dans ces deux situations.

#### 4.1.1 Minimisation du risque et régression

Nous commençons par discuter la minimisation du risque moyen de prédiction  $R_p$  pour les problèmes de régression.

##### 4.1.1.1 Régression et coût quadratique

Dans le cas de la régression, l'étiquette  $y$  prend des valeurs scalaires ou vectorielles. Un coût classique est le coût quadratique :

$$\ell(y, h(\mathbf{x})) = \|y - h(\mathbf{x})\|_2^2$$

Pour alléger les notations, on omettra l'indice 2 de la norme dans la suite du chapitre : il s'agira toujours de la norme euclidienne.

En développant  $\|y - h(\mathbf{x})\|^2 = \|y\|^2 + \|h(\mathbf{x})\|^2 - 2y \cdot h(\mathbf{x})$  :

$$\begin{aligned} R_p(h) &= E_{(\mathbf{X}, Y)}(\ell(Y, h(\mathbf{X}))) \\ &= \iint \|y\|^2 p(\mathbf{x}, y) \, d\mathbf{x} \, dy + \iint (\|h(\mathbf{x})\|^2 - 2y \cdot h(\mathbf{x})) p(\mathbf{x}, y) \, d\mathbf{x} \, dy \end{aligned}$$

Comme la première intégrale ne dépend pas de  $h$ , minimiser  $R_p$  revient à minimiser la seconde intégrale. Or, en écrivant  $p(\mathbf{x}, y) = p(\mathbf{x})p(y|\mathbf{x})$  :

$$\begin{aligned} &\iint (\|h(\mathbf{x})\|^2 - 2y \cdot h(\mathbf{x})) p(\mathbf{x}, y) \, d\mathbf{x} \, dy \\ &= \iint \|h(\mathbf{x})\|^2 p(\mathbf{x})p(y|\mathbf{x}) \, d\mathbf{x} \, dy - 2 \int \left( \int y p(y|\mathbf{x}) \, dy \right) \cdot h(\mathbf{x}) p(\mathbf{x}) \, d\mathbf{x} \end{aligned}$$

Comme  $\iint \|h(\mathbf{x})\|^2 p(\mathbf{x})p(y|\mathbf{x}) \, d\mathbf{x} \, dy = \int \|h(\mathbf{x})\|^2 p(\mathbf{x}) \left( \int p(y|\mathbf{x}) \, dy \right) \, d\mathbf{x}$  et  $\int p(y|\mathbf{x}) \, dy = 1$ , minimiser  $R_p$  revient à minimiser :

$$\begin{aligned} &\int \|h(\mathbf{x})\|^2 p(\mathbf{x}) \, d\mathbf{x} - 2 \int \left( \int y p(y|\mathbf{x}) \, dy \right) \cdot h(\mathbf{x}) p(\mathbf{x}) \, d\mathbf{x} \\ &= \int \left( \|h(\mathbf{x})\|^2 - 2 \left( \int y p(y|\mathbf{x}) \, dy \right) \cdot h(\mathbf{x}) \right) p(\mathbf{x}) \, d\mathbf{x} \end{aligned}$$

Néanmoins,

$$\|h(\mathbf{x})\|^2 - 2 \left( \int y p(y|\mathbf{x}) \, dy \right) \cdot h(\mathbf{x}) = \left\| h(\mathbf{x}) - \int y p(y|\mathbf{x}) \, dy \right\|^2 - \left\| \int y p(y|\mathbf{x}) \, dy \right\|^2$$

Le second terme ne dépendant pas de  $h$ , on peut conclure : minimiser  $R_p(h)$  par rapport à  $h$  revient à minimiser

$$\int \left\| h(\mathbf{x}) - \int y p(y|\mathbf{x}) \, dy \right\|^2 p(\mathbf{x}) \, d\mathbf{x}$$



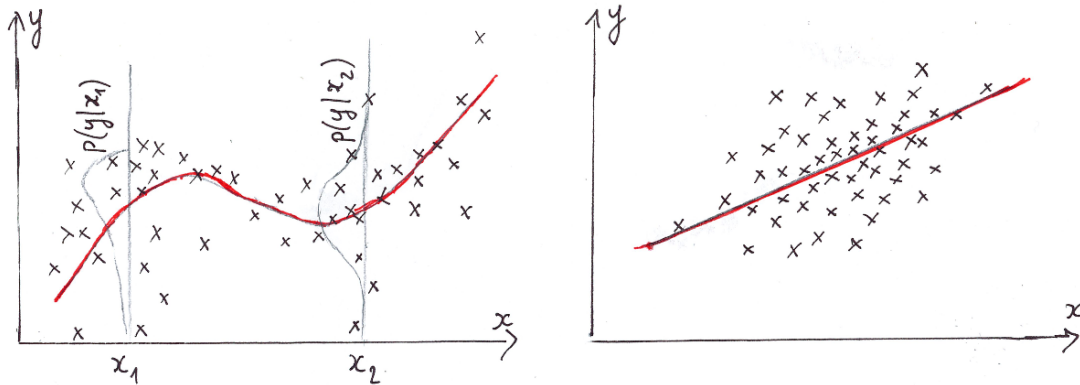


FIGURE 4.1 – Régression pour un coût quadratique. Dans ces exemples, on souhaite prédire  $y \in \mathbb{R}$  en fonction de l'observation  $x \in \mathbb{R}$ . L'estimateur optimal est représenté en trait plein et est donné en chaque  $x$  par :  $h(x) = \int yp(y|x) dy = E(Y|X = x)$ . À gauche : situation générale dans laquelle la forme de la densité  $p(y|x)$  varie selon  $x$ . À droite : lorsque  $(X, Y)$  est un vecteur gaussien,  $h$  est un estimateur linéaire.

c'est-à-dire choisir :

$$h(\mathbf{x}) = \int yp(y|\mathbf{x}) dy$$

Cette dernière quantité étant l'espérance conditionnelle de  $Y$  sachant  $\mathbf{X} = \mathbf{x}$ , on peut aussi écrire :  $h(\mathbf{x}) = E(Y|\mathbf{X} = \mathbf{x})$ . De même,  $p(y|\mathbf{x}) = p(\mathbf{x}, y)/p(\mathbf{x})$  est la densité conditionnelle de  $Y$  sachant  $\mathbf{X} = \mathbf{x}$ .

Nous avons donc démontré la proposition suivante.

**Proposition 4.1** *Le régresseur minimisant le risque moyen de prédiction pour un coût d'erreur quadratique est donné par la relation :*

$$h(\mathbf{x}) = E(Y|\mathbf{X} = \mathbf{x}) = \int yp(y|\mathbf{x}) dy$$

#### 4.1.1.2 En pratique

En pratique, on ne connaît ni  $p(\mathbf{x}, y)$  ni  $p(y|\mathbf{x})$ , mais seulement un ensemble fini d'observations étiquetées  $(\mathbf{x}_n, y_n)_{1 \leq n \leq N}$ . En supposant que ce jeu de données a permis d'estimer la densité conjointe  $p(\mathbf{x}, y)$ , on peut alors déterminer le prédicteur  $h(\mathbf{x}) = \int yp(y|\mathbf{x}) dy$ , comme illustré par la figure 4.1

Il existe des cas où on peut déterminer  $h(\mathbf{x})$  plus explicitement. Par exemple, supposons que  $p(\mathbf{x}, y)$  est une densité gaussienne. Nous disposons du théorème suivant (admis) :

**Théorème 4.1** Si  $(X, Y)$  est un vecteur aléatoire gaussien de moyenne  $(E(X), E(Y))$  et de matrice de covariance  $\begin{pmatrix} \Sigma_X & \Sigma_0 \\ \Sigma_0^T & \Sigma_Y \end{pmatrix}$  (de manière à ce que  $\Sigma_X$  soit la matrice de covariance de  $X$ ,  $\Sigma_Y$  celle de  $Y$ , et  $\Sigma_0$  la matrice de covariance de  $X$  et  $Y$ ), alors la loi conditionnelle  $p(y|x)$  est une loi gaussienne d'espérance :

$$E(Y|X = \mathbf{x}) = E(Y) + \Sigma_0^T \Sigma_X^{-1} (\mathbf{x} - E(X))$$

Nous déduisons donc :

**Proposition 4.2** Dans le cas où  $p(\mathbf{x}, y)$  est une densité gaussienne, l'estimateur  $h$  minimisant le risque moyen de prédiction pour un coût quadratique est une fonction affine des observations.

Cet estimateur  $h(\mathbf{x})$  s'exprime, par l'intermédiaire de la formule précédente, en fonction de  $E(X)$ ,  $E(Y)$ ,  $\Sigma_0$  et  $\Sigma_X$ .

Bien entendu, ces quantités sont généralement inconnues. On remplace donc en pratique la minimisation du risque de prédiction moyen par la minimisation du risque empirique. Dans le cas d'une loi jointe de  $(X, Y)$  gaussienne, comme on cherche une fonction de régression affine, on se retrouve dans la situation de l'estimation des paramètres de la régression linéaire au sens des moindres carrés des résidus, rappelée au chapitre 1, section 1.4.2, et étudiée dans les cours de statistique.

Sans hypothèse sur la loi jointe de  $(X, Y)$ , on peut faire une estimation de  $E(Y|X = \mathbf{x})$  à partir des observations étiquetées. Le régresseur optimal est  $h(\mathbf{x}) = \int y p(y|\mathbf{x}) dy$ , ce qui s'écrit :

$$h(\mathbf{x}) = \int y \frac{p(\mathbf{x}, y)}{p(\mathbf{x})} dy$$

Le chapitre 5, plus particulièrement la section 5.1.2, présenteront l'estimation des densités de probabilité par la méthode des fenêtres de Parzen. Avec une fenêtre (ou noyau), dépendant d'une largeur de bande  $h$  (avec les notations de la section 5.1.2,  $W_h = \frac{1}{h^d} W(\mathbf{x}/h)$ ), on peut estimer la loi jointe  $p(\mathbf{x}, y)$  à partir des observations étiquetées par :

$$p(\mathbf{x}, y) \approx \frac{1}{N} \sum_{i=1}^N W_h(\mathbf{x} - \mathbf{x}_i) W_h(y - y_i)$$

et  $p(\mathbf{x})$  par :

$$p(\mathbf{x}) \approx \frac{1}{N} \sum_{i=1}^N W_h(\mathbf{x} - \mathbf{x}_i)$$

Ainsi :

$$\begin{aligned} \int y \frac{p(\mathbf{x}, y)}{p(\mathbf{x})} dy &\approx \int y \frac{\sum_{i=1}^N W_h(\mathbf{x} - \mathbf{x}_i) W_h(y - y_i)}{\sum_{i=1}^N W_h(\mathbf{x} - \mathbf{x}_i)} dy \\ &= \frac{\sum_{i=1}^N W_h(\mathbf{x} - \mathbf{x}_i) \int y W_h(y - y_i) dy}{\sum_{i=1}^N W_h(\mathbf{x} - \mathbf{x}_i)} \\ &= \frac{\sum_{i=1}^N y_i W_h(\mathbf{x} - \mathbf{x}_i)}{\sum_{i=1}^N W_h(\mathbf{x} - \mathbf{x}_i)} \end{aligned}$$

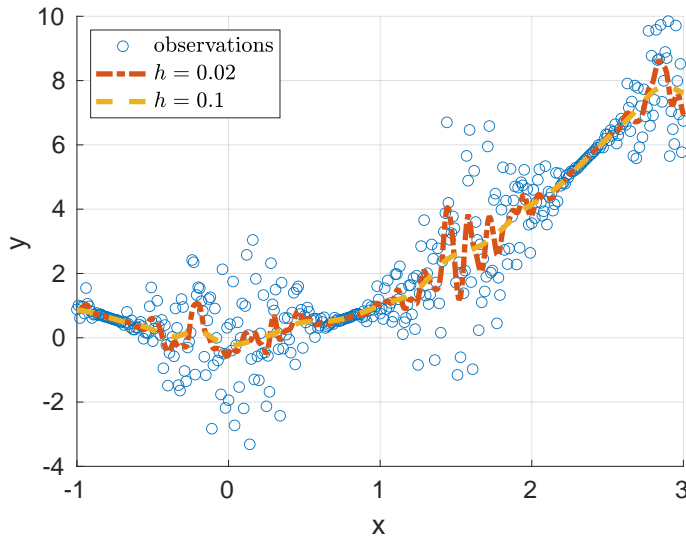


FIGURE 4.2 – Régression à noyau de Nadaraya-Watson. Le noyau est ici une fonction gaussienne d'écart-type  $h$ .

En effet,  $\int y W_h(y - y_i) dy = y_i \int W_h(y) dy + \int y W_h(y) dy = y_i$  car  $\int W_h(y) dy = 1$  dans la méthode de Parzen, et  $W_h$  étant une fonction paire,  $\int y W_h(y) dy = 0$ .

On obtient donc, à partir des observations étiquetées  $(\mathbf{x}_i, y_i)_{1 \leq i \leq N}$ , la fonction de régression suivante :

$$h(\mathbf{x}) = \frac{\sum_{i=1}^N y_i W_h(\mathbf{x} - \mathbf{x}_i)}{\sum_{i=1}^N W_h(\mathbf{x} - \mathbf{x}_i)}$$

Il s'agit de l'estimateur de Nadaraya-Watson. On parle aussi de régression par noyau (*kernel regression*) de Nadaraya-Watson. Pour calculer  $h(\mathbf{x})$ , on fait la moyenne pondérée des  $y_i$ , les poids étant fonction de la distance entre  $\mathbf{x}$  et  $\mathbf{x}_i$ . Cette méthode fait partie des méthodes de régression non-paramétrique, dans lesquelles la régression n'est pas gouvernée par des paramètres à estimer à partir des données, contrairement à la régression linéaire par exemple. Le seul paramètre est la largeur de bande  $h$ , fixée par l'utilisateur. Remarquons que, comme tout hyperparamètre,  $h$  peut être choisi par validation croisée. Voir la figure 4.2.

Attention, le noyau et la régression à noyau évoquée ci-dessus n'ont aucun rapport avec les méthodes à noyau discutées au chapitre 8.

#### 4.1.2 Minimisation du risque et classification

Nous discutons à présent la minimisation du risque moyen de prédiction  $R_p$  pour les problèmes de classification.

#### 4.1.2.1 Classification et coût 0-1

Dans le cas de la classification supervisée, l'étiquette  $y$  prend une valeur correspondant à une classe parmi  $K$ . Les classes forment une partition des observations. Nous noterons  $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_K$  les classes, et  $y = \mathcal{C}_1, y = \mathcal{C}_2$ , etc. les valeurs prises par  $y$ .

Une fonction de coût classique pour les problèmes de classification est le coût 0-1 :

$$\ell(y, h(\mathbf{x})) = \begin{cases} 0 & \text{si } h(\mathbf{x}) = y \quad (\text{pas d'erreur de classification}) \\ 1 & \text{si } h(\mathbf{x}) \neq y \quad (\text{erreur de classification}) \end{cases}$$

On écrit plus succinctement  $\ell(y, h(\mathbf{x})) = \mathbb{1}_F(\mathbf{x}, y)$ , où  $\mathbb{1}_F$  est la fonction indicatrice du sous-ensemble  $F = \{(\mathbf{x}, y), h(\mathbf{x}) \neq y\}$  de l'ensemble des observations étiquetées. Autrement dit,  $F$  est l'ensemble des observations étiquetées dont la classe n'est pas correctement prédite par  $h$ . Rappelons que la fonction indicatrice d'un ensemble vaut 1 sur cet ensemble et 0 en dehors.

#### 4.1.2.2 Notations et vocabulaire

Les distributions de probabilités suivantes vont nous être utiles.

- $p(\mathcal{C}_k)$  est la probabilité a priori (*prior probability*) de chaque classe. Bien entendu,  $\sum_{k=1}^K p(\mathcal{C}_k) = 1$ . Il s'agit de l'information potentiellement connue avant d'avoir observé des données.

Par exemple, dans un problème de reconnaissance de caractères, le but est d'associer des images figurant les caractères au caractère représenté. On peut connaître la probabilité a priori d'un caractère à partir des fréquences d'apparition dans la langue du document :  $p(a) = 0.07$ ,  $p(b) = 0.01$ ,  $p(c) = 0.03$ , etc.

- $p(\mathbf{x}|\mathcal{C}_k)$  est la probabilité conditionnelle. On devrait parler en fait de densité (ou vraisemblance, *likelihood*) mais bon nombre d'auteurs parlent abusivement de probabilité dans la littérature de l'apprentissage automatique. Il s'agit de la densité de la loi de probabilité d'une observation tirée dans la classe  $\mathcal{C}_k$ .

Un choix classique est le modèle gaussien. Dans le cas où  $\mathbf{x} \in \mathbb{R}^d$ ,

$$p(\mathbf{x}|\mathcal{C}_k) = \frac{1}{(2\pi)^{d/2} |\Sigma_k|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T \Sigma_k^{-1} (\mathbf{x}-\boldsymbol{\mu}_k)}$$

la classe  $\mathcal{C}_k$  étant caractérisée par la moyenne  $\boldsymbol{\mu}_k$  et la matrice de covariance  $\Sigma_k$  de déterminant  $|\Sigma_k|$ .

Pour reprendre l'exemple de la reconnaissance de caractères : lorsqu'une image (de taille, disons,  $16 \times 16$  pixels) du caractère  $a$  est représentée par un vecteur  $\mathbf{x}$  (de taille 256), la classe  $a$  est caractérisé par un « a moyen »  $\boldsymbol{\mu}_a$  et une matrice  $\Sigma_a$  quantifiant les variations autour de la moyenne.

- $p(\mathcal{C}_k|\mathbf{x})$  est la probabilité a posteriori (*posterior probability*).

Il s'agit de la probabilité de la classe  $\mathcal{C}_k$  étant donnée l'observation  $\mathbf{x}$ . Comme nous allons le voir, nous chercherons à déterminer cette probabilité a posteriori.

Ces probabilités sont liées par le théorème de Bayes :

$$\forall 1 \leq k \leq K, p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})}$$

et par la formule des probabilités totales, les classes  $\mathcal{C}_k$  étant disjointes :

$$p(\mathbf{x}) = \sum_{k=1}^K p(\mathbf{x}, \mathcal{C}_k) = \sum_{k=1}^K p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)$$

#### 4.1.2.3 Classifieur de Bayes

Pour simplifier les notations, nous nous restreignons au cas biclasse ( $K = 2$ ) dans la suite du chapitre.

Avec un coût 0-1, le risque de prédiction s'écrit :

$$\begin{aligned} R_p(h) &= E_{(X,Y)}(\ell(Y, h(\mathbf{X}))) = E_{(X,Y)}(\mathbb{1}_F(\mathbf{X}, Y)) \\ &= \iint \mathbb{1}_F(\mathbf{x}, y) p(\mathbf{x}, y) \, d\mathbf{x} \, dy \end{aligned}$$

De plus, l'ensemble  $F$  (des couples  $(\mathbf{x}, y)$  pour lesquels la prédiction  $h(\mathbf{x})$  est différente de  $y$ , donc fausse) s'écrit comme une union disjointe :

$$F = F_1 \cup F_2$$

où

$$F_1 = \{(\mathbf{x}, y) \text{ tel que } h(\mathbf{x}) = \mathcal{C}_1 \text{ et } y = \mathcal{C}_2\}$$

(couples mal classés car la prédiction est  $\mathcal{C}_1$  alors que la vraie classe est  $\mathcal{C}_2$ ), et

$$F_2 = \{(\mathbf{x}, y) \text{ tel que } h(\mathbf{x}) = \mathcal{C}_2 \text{ et } y = \mathcal{C}_1\}$$

(couples mal classés car la prédiction est  $\mathcal{C}_2$  alors que la vraie classe est  $\mathcal{C}_1$ ).

Ainsi,

$$\begin{aligned} R_p(h) &= \int \mathbb{1}_{F_1}(\mathbf{x}, \mathcal{C}_2) p(\mathbf{x}, \mathcal{C}_2) \, d\mathbf{x} + \int \mathbb{1}_{F_2}(\mathbf{x}, \mathcal{C}_1) p(\mathbf{x}, \mathcal{C}_1) \, d\mathbf{x} \\ &= \int_{\mathcal{R}_1} p(\mathbf{x}, \mathcal{C}_2) \, d\mathbf{x} + \int_{\mathcal{R}_2} p(\mathbf{x}, \mathcal{C}_1) \, d\mathbf{x} \end{aligned}$$

où

$$\mathcal{R}_1 = \{\mathbf{x} \text{ tel que } h(\mathbf{x}) = \mathcal{C}_1\}$$

et

$$\mathcal{R}_2 = \{\mathbf{x} \text{ tel que } h(\mathbf{x}) = \mathcal{C}_2\}$$

Ces deux ensembles séparent les observations respectivement en celles classées dans  $\mathcal{C}_1$  par  $h$  et celles classées dans  $\mathcal{C}_2$ .

Remarquons que si  $k = 1$  ou  $k = 2$ ,  $p(\mathbf{x}, \mathcal{C}_k) = p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k) = p(\mathcal{C}_k|\mathbf{x})p(\mathbf{x})$ .

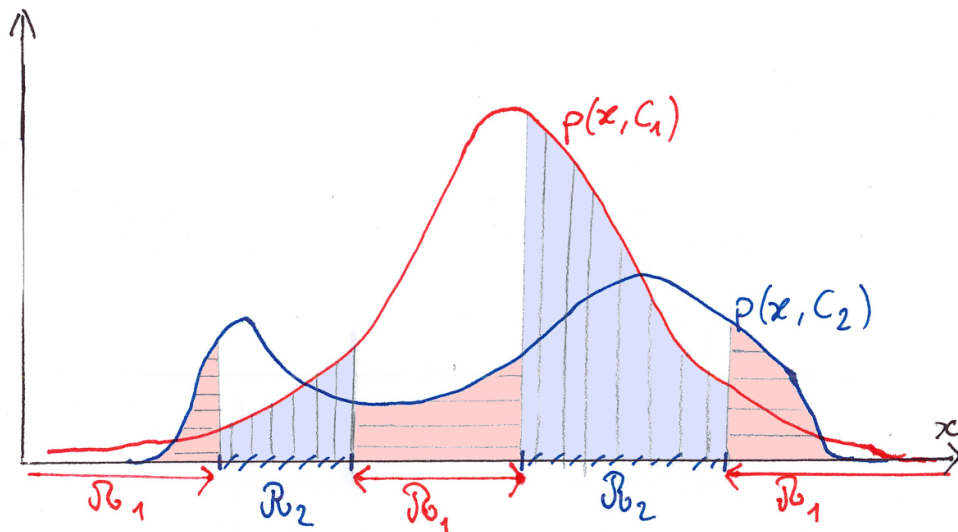


FIGURE 4.3 – Minimisation du risque de prédiction. Nous représentons les densités  $p(\mathbf{x}, \mathcal{C}_1)$  et  $p(\mathbf{x}, \mathcal{C}_2)$  dans le cas où  $\mathbf{x} \in \mathbb{R}$  (les observations sont des scalaires). Les ensembles  $\mathcal{R}_1$  et  $\mathcal{R}_2$  réalisent une partition de l'espace des observations. Le risque de prédiction  $R_p(h) = \int_{\mathcal{R}_1} p(\mathbf{x}, \mathcal{C}_2) dx + \int_{\mathcal{R}_2} p(\mathbf{x}, \mathcal{C}_1) dx$  apparaît comme la somme des surfaces hachurées horizontalement (première intégrale) et verticalement (seconde intégrale). Pour minimiser  $R_p$ , il faut donc choisir  $\mathcal{R}_1$  comme l'ensemble des  $\mathbf{x}$  tels que  $p(\mathbf{x}, \mathcal{C}_2) < p(\mathbf{x}, \mathcal{C}_1)$  et  $\mathcal{R}_2$  comme l'ensemble des  $\mathbf{x}$  tels que  $p(\mathbf{x}, \mathcal{C}_1) < p(\mathbf{x}, \mathcal{C}_2)$ . Cela se résume en la règle de décision  $h(\mathbf{x}) = \operatorname{argmax}_{\mathcal{C}_k} p(\mathbf{x}, \mathcal{C}_k)$ . Ce raisonnement s'étend à des observations non scalaires et au cas de  $K > 2$  classes.

Le problème est de choisir  $h$  de manière à minimiser  $R_p(h)$ . L'illustration de la figure 4.3 permet de répondre à la question : il faut que  $h(\mathbf{x})$  soit la classe  $\mathcal{C}_1$  si  $p(\mathbf{x}, \mathcal{C}_1) > p(\mathbf{x}, \mathcal{C}_2)$ , et la classe  $\mathcal{C}_2$  dans le cas contraire. Le classifieur optimal  $h$  vérifie donc :  $h(\mathbf{x}) = \operatorname{argmax}_{\mathcal{C}_k} p(\mathbf{x}, \mathcal{C}_k)$ . On conclut avec la remarque précédente liant  $p(\mathbf{x}, \mathcal{C}_k)$ , vraisemblance  $p(\mathbf{x}|\mathcal{C}_k)$ , et probabilités a posteriori  $p(\mathcal{C}_k|\mathbf{x})$  et a priori  $p(\mathcal{C}_k)$  :

**Proposition 4.3** Le classifieur minimisant le risque moyen de prédiction pour un coût d'erreur 0-1 est défini par :

$$h(\mathbf{x}) = \operatorname{argmax}_{\mathcal{C}_k} p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)$$

ou, de manière équivalente :

$$h(\mathbf{x}) = \operatorname{argmax}_{\mathcal{C}_k} p(\mathcal{C}_k|\mathbf{x})$$

Ce classifieur est appelé classifieur de Bayes. Comme il consiste à associer  $\mathbf{x}$  à la classe de probabilité a posteriori maximale, on l'appelle aussi classifieur du maximum a posteriori (MAP).

On remarque que le classifieur de Bayes peut classer une observation  $\mathbf{x}$  dans une classe  $\mathcal{C}_1$ , même si  $p(\mathbf{x}|\mathcal{C}_1)$  n'est pas la vraisemblance la plus élevée parmi les  $p(\mathbf{x}|\mathcal{C}_k)$ , à condition que la probabilité a priori  $p(\mathcal{C}_1)$  ait une valeur élevée.

Comme dans la discussion sur la régression, il ne faut pas perdre de vue que le raisonnement a été mené en supposant les différentes distributions de probabilités connues. Le classifieur de Bayes est le classifieur idéal dans le cas du risque 0-1, dans le sens où il réalise le minimum du risque de prédiction (ce minimum s'appelle d'ailleurs « risque de Bayes »). Néanmoins, pour le mettre en œuvre il faut être capable d'estimer les distributions de probabilité de manière fiable (voir le chapitre 5), ou d'imposer des hypothèses additionnelles, comme nous le ferons dans le chapitre 6.

## 4.2 Pour approfondir...

**Régression robuste, M-estimateur, RANSAC** Comme mentionné précédemment, l'inconvénient de la régression avec un coût quadratique est l'influence de certaines observations : ce coût donne un poids très fort à des données aberrantes puisqu'elles contribuent comme le carré du résidu  $\|y_n - f_{\boldsymbol{\theta}}(\mathbf{x}_n)\|_2$ , qui est grand dans ce cas. Plutôt que minimiser la somme des carrés des résidus

$$\sum_{n=1}^N \|y_n - f_{\boldsymbol{\theta}}(\mathbf{x}_n)\|_2^2$$

en fonction des paramètres  $\boldsymbol{\theta}$  du modèle, un M-estimateur cherchera à minimiser à l'aide d'un algorithme d'optimisation numérique :

$$\sum_{n=1}^N \rho(y_n - f_{\boldsymbol{\theta}}(\mathbf{x}_n))$$

avec  $\rho$  une fonction donnant un coût moins grand aux données aberrantes, de résidus élevés. Par exemple, le coût de Huber (*Huber loss*) s'exprime comme :

$$\rho(x) = \begin{cases} x^2 & \text{si } |x| \leq \delta \\ 2\delta|x| - \delta^2 & \text{sinon} \end{cases}$$

où  $\delta$  est un paramètre positif. Le coût de Huber a l'avantage de croître linéairement plutôt que quadratiquement, comme illustré par la figure 4.4. Cela diminue l'influence des données aberrantes.

Dans le même esprit, on peut utiliser :

$$\rho(x) = \begin{cases} 0 & \text{si } |x| \leq \delta \\ 1 & \text{sinon} \end{cases}$$

Les observations vérifiant  $|y_n - f_{\boldsymbol{\theta}}(\mathbf{x}_n)| \leq \delta$  forment ce qu'on appelle l'« ensemble de consensus ». La quantité  $N - \sum_n \rho(y_n - f_{\boldsymbol{\theta}}(\mathbf{x}_n))$  est le cardinal de cet ensemble. L'objectif de minimiser  $\sum_n \rho(y_n - f_{\boldsymbol{\theta}}(\mathbf{x}_n))$  est donc équivalent à celui de maximiser le cardinal de l'ensemble

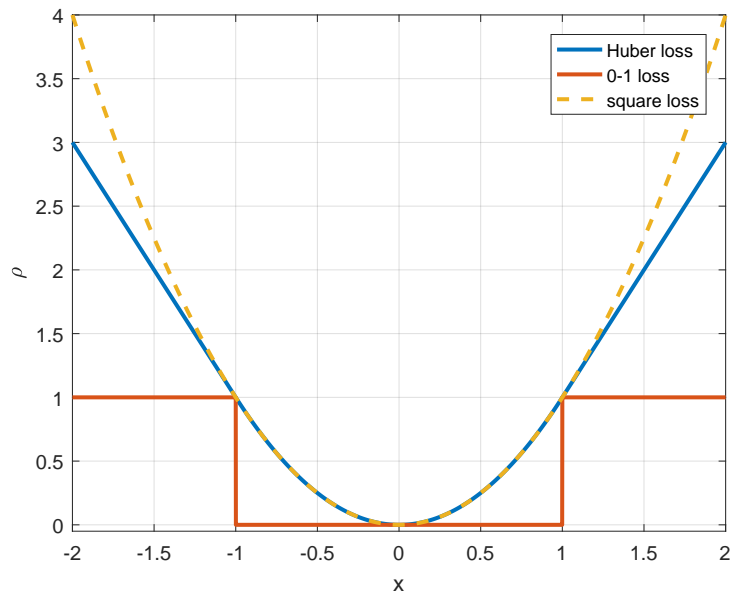


FIGURE 4.4 – Croissance comparée des coûts de Huber ( $\delta = 1$ ), 0-1, et quadratique.

de consensus. La fonction  $\rho$  n'étant ni dérivable, ni convexe, on ne peut plus utiliser d'algorithme d'optimisation numérique basé sur le gradient (par exemple); il faut utiliser un algorithme d'optimisation combinatoire. C'est ce que réalise le célèbre algorithme RANSAC (Random Sample Consensus, Fischler et Bolles 1981). RANSAC est très utile dans des situations où une grande proportion de données aberrantes sont présentes, comme dans les problèmes de vision par ordinateur. Le principe est relativement simple et vous êtes invité à faire une recherche Internet si vous voulez en savoir davantage.



# Estimation de densités de probabilité

Le classifieur de Bayes et le régresseur optimal définis au chapitre 4 reposent sur la connaissance de différentes lois de probabilités.

Dans le problème de la classification supervisée, il faut avoir une connaissance des probabilités a priori  $p(\mathcal{C}_k)$ . Les approches les plus populaires sont les suivantes.

- On dispose d'une information, indépendamment des observations. C'est le cas par exemple de la reconnaissance de caractères discutée au paragraphe 4.1.2.2.
- Les probabilités  $p(\mathcal{C}_k)$  sont les fréquences estimées sur la base des observations étiquetées :

$$p(\mathcal{C}_k) = \frac{\#\{\mathbf{x}_n \in \mathcal{C}_k, 1 \leq n \leq N\}}{N}$$

où # désigne le cardinal d'un ensemble fini.

- On peut aussi juger qu'aucune classe n'est vraiment prépondérante, et supposer les probabilités a priori  $p(\mathcal{C}_k)$  toutes égales. Dans ce cas, le classifieur de Bayes se simplifie en  $h(\mathbf{x}) = \operatorname{argmax}_{\mathcal{C}_k} p(\mathbf{x}|\mathcal{C}_k)$ . Il s'agit de la règle du maximum de vraisemblance (*maximum of likelihood*, ML).

En ce qui concerne les densités  $p(\mathbf{x}|\mathcal{C}_k)$  (cas de la classification) et  $p(y|\mathbf{x})$  (cas de la régression), il faut les estimer à partir des données étiquetées. On utilisera alors les méthodes générales d'estimation de densités de probabilité, qui est une branche de l'apprentissage non-supervisé. Le fait que dans le problème initial les observations sont étiquetées ne change pas grand-chose.

Dans ce chapitre, on suppose disposer d'un jeu de données de  $N$  observations  $(\mathbf{x}_n)_{1 \leq n \leq N}$  appartenant à un espace vectoriel  $\mathbb{R}^d$ , et on cherche à estimer la densité de probabilité sous-jacente. On distingue les méthodes non-paramétriques et les méthodes paramétriques.

## 5.1 Méthodes non-paramétriques

Ces méthodes sont dites non-paramétriques dans le sens où la densité estimée ne dépend pas directement d'un ou plusieurs paramètres.

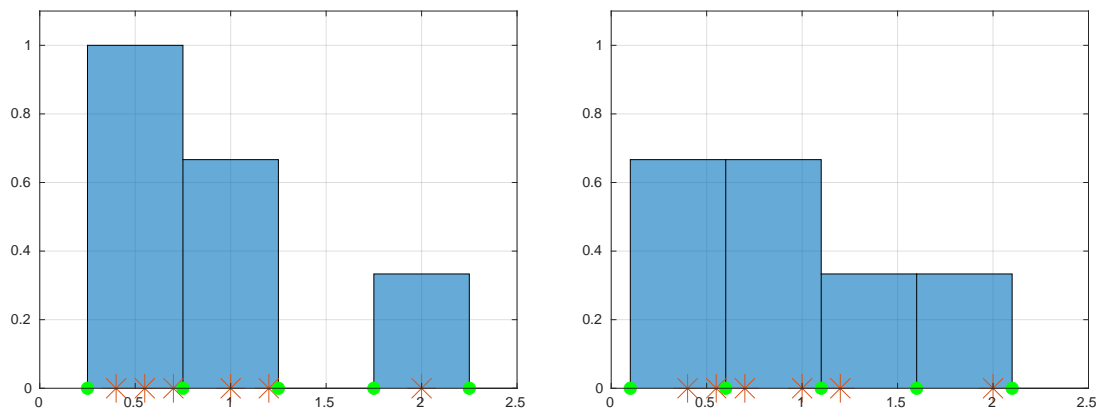


FIGURE 5.1 – Estimation de densité par histogramme. Dans cet exemple, les observations (les mêmes dans les deux cas) sont des réels et sont représentées par des croix sur l'axe des abscisses. Les histogrammes sont définis comme la proportion d'observations « tombant » dans les baquets (des intervalles dont les limites sont marquées par un petit disque), normalisée de manière à ce que l'intégrale sous l'histogramme soit égale à 1. On constate que la fonction constante par morceaux censée approcher la densité dépend fortement du choix de la discrétisation de l'espace des observations (ici une légère translation de la position des « baquets » change drastiquement l'aspect des histogrammes, mais un changement de largeur aurait aussi pu avoir un impact important).

### 5.1.1 Histogrammes

Pour estimer une densité de probabilité par histogramme, on commence par définir une discrétisation de l'espace des observations pour construire des « baquets » (*bins*), puis on calcule la proportion des observations tombant dans chaque baquet. La densité de probabilité est estimée comme la proportion divisée par la mesure du baquet, de manière à ce que l'intégrale de la densité estimée sur l'espace soit égale à 1. La densité estimée est donc constante sur chaque baquet.

Dans le cas d'observations scalaires ( $d = 1$ ), les baquets sont des intervalles, et en dimension plus grande des pavés.

Dans cette méthode, le choix de la discrétisation est parfois critique, comme l'illustre la figure 5.1.

Lorsqu'on souhaite une estimation plus régulière qu'une fonction constante par morceaux, on utilise l'estimation par fenêtre de Parzen.

### 5.1.2 Méthode de Parzen

L'estimation est faite en centrant une fonction (obtenue à partir d'une fenêtre, ou noyau, notée  $W$ ) sur chaque observation et en moyennant les fonctions sur l'ensemble des observa-

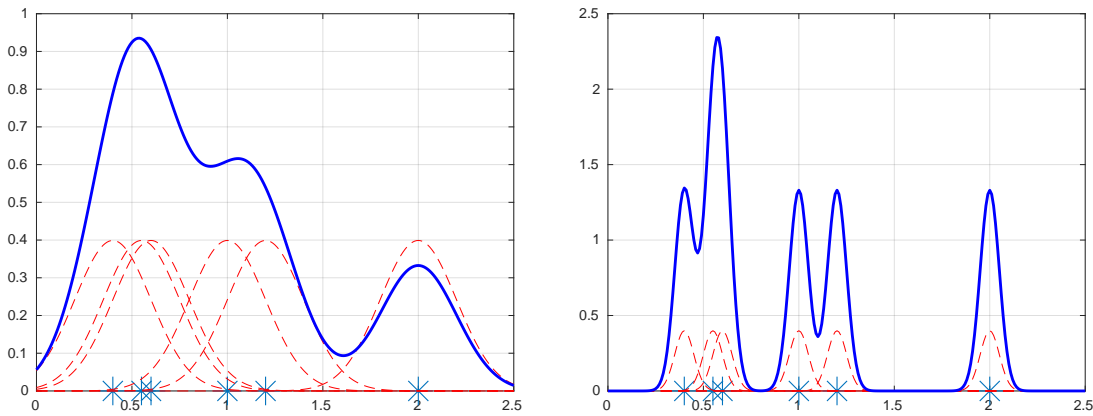


FIGURE 5.2 – Estimation de densité par la méthode de Parzen : ici, fenêtre gaussienne. À gauche :  $h = 0.2$  ; à droite :  $h = 0.05$ . En bleu : densité estimée ; en rouge :  $W((x - x_n)/h)$  pour chaque  $x_n$ . Une largeur de bande  $h$  trop petite fait apparaître des détails indésirables, et une largeur trop grande lisse des détails pertinents.

tions. Autrement dit, l'estimation en un point  $\mathbf{x}$  est :

$$p_W(\mathbf{x}) = \frac{1}{Nh^d} \sum_{n=1}^N W\left(\frac{\mathbf{x} - \mathbf{x}_n}{h}\right)$$

où  $W$  est une fonction concentrée autour de 0, d'intégrale 1 sur l'espace des observations, et où  $h > 0$  est appelée « largeur de bande » (*bandwidth*).

L'estimation  $p_W$  a la même régularité que  $W$  : si  $W$  est de classe  $C^\infty$ , alors  $p_W$  aussi.

Le paramètre  $h$  gouverne l'influence de chaque observation  $\mathbf{x}_n$  : si  $h$  est grand,  $p_W(\mathbf{x})$  dépendra de  $\mathbf{x}_n$  même si  $\mathbf{x}$  est éloigné de  $\mathbf{x}_n$ .

La normalisation  $1/(Nh^d)$  est telle que l'intégrale de  $p_W$  sur l'espace des observations est 1 (ce qui est logique pour une densité).

On parle d'estimation par noyaux ou par fenêtres de Parzen, ce qu'illustre la figure 5.2.

Un choix classique de  $W$  est la fenêtre gaussienne :

$$W(\mathbf{x}) = W_0 \exp(-\|\mathbf{x}\|_2^2/2)$$

avec  $W_0$  fixé de manière à ce que  $\int W = 1$ .

L'inconvénient de cette méthode est le choix de la largeur de bande  $h$  laissé à l'utilisateur alors qu'il a une influence forte sur l'estimation de la densité.

### 5.1.3 Méthodes des $K$ plus proches voisins

Pour estimer une densité  $\phi$  en  $\mathbf{x}$ , on considère une boule  $B_{\mathbf{x}}$  centrée en  $\mathbf{x}$ , contenant  $K$  observations parmi les  $N$  disponibles en tout qui suivent la loi  $\phi$ .

D'après la loi des grands nombres (écrire l'espérance de la fonction indicatrice de la boule  $B_x$ ) :

$$\int_{B_x} \phi(\mathbf{y}) \, d\mathbf{y} \simeq \frac{K}{N}$$

Par ailleurs, si on suppose la densité  $\phi$  à peu près constante sur  $B_x$  :

$$\int_{B_x} \phi(\mathbf{y}) \, d\mathbf{y} \simeq \phi(\mathbf{x}) \times \text{Volume}(B_x)$$

Donc :

$$\frac{K}{N} \simeq \int_{B_x} \phi(\mathbf{y}) \, d\mathbf{y} \simeq \phi(\mathbf{x}) \times \text{Volume}(B_x)$$

On en déduit l'estimateur des  $K$  plus proches voisins :

$$\hat{\phi}(\mathbf{x}) = \frac{K}{N V_K(\mathbf{x})}$$

où  $V_K(\mathbf{x})$  est le volume d'une boule centrée en  $\mathbf{x}$  contenant les  $K$  plus proches voisins de  $\mathbf{x}$ , sur laquelle la densité est supposée constante.

En fait, cette méthode est rarement utilisée pour estimer des densités, mais elle apparaîtra dans la méthode des  $K$  plus proches voisins pour la classification supervisée au chapitre 6.

## 5.2 Méthodes paramétriques

On cherche la distribution dans une famille paramétrée. Par exemple, la distribution d'une variable aléatoire gaussienne est caractérisée par la moyenne et la variance dont on connaît des estimateurs non-biaisés d'après le cours de statistique :

$$\boldsymbol{\mu} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n \quad \text{et} \quad \sigma^2 = \frac{1}{N-1} \sum_{n=1}^N (\mathbf{x}_n - \boldsymbol{\mu})^2$$

Rappelons que pour un vecteur aléatoire gaussien, des estimateurs de la moyenne (qui est alors un vecteur)  $\boldsymbol{\mu}$  et de la matrice de covariance  $\Sigma$  sont :

$$\boldsymbol{\mu} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n \quad \text{et} \quad \Sigma = \frac{1}{N-1} \sum_{n=1}^N (\mathbf{x}_n - \boldsymbol{\mu})(\mathbf{x}_n - \boldsymbol{\mu})^T$$

Le paragraphe 5.3 discute les problèmes pratiques posés en grande dimension par l'estimateur  $\Sigma$ .

De manière générale, dans les méthodes paramétriques, on suppose la densité de la forme  $f_{\boldsymbol{\theta}}$  où  $\boldsymbol{\theta}$  représente un ou plusieurs paramètres. La méthode du maximum de vraisemblance (Fisher 1922) permet alors d'estimer  $\boldsymbol{\theta}$  à partir des observations.

### 5.2.1 Estimateur du maximum de vraisemblance

Attention, ici l'expression « maximum de vraisemblance » a un sens différent de celui du chapitre 4.

On définit la vraisemblance d'un paramètre  $\theta$ , notée  $\mathcal{L}(\theta)$ , comme la valeur de la densité de probabilité évaluée en les observations. Si on suppose les observations indépendantes et identiquement distribuées selon une loi  $f_\theta$ , comme la loi jointe est le produit des lois marginales, on peut écrire :

$$\mathcal{L}(\theta) = \prod_{i=1}^N f_\theta(x_i)$$

La vraisemblance « décrit la plausibilité d'une valeur des paramètres d'un modèle, étant donné l'observation d'un certain nombre de réalisations d'une variable aléatoire »<sup>1</sup>.

Pour simplifier les calculs, il peut être utile de considérer la log-vraisemblance (le logarithme de la vraisemblance), notée  $L(\theta)$ , telle que  $L(\theta) = \log(\mathcal{L}(\theta)) = \sum_{i=1}^N \log(f_\theta(x_i))$ .

La méthode du maximum de vraisemblance consiste alors à chercher  $\theta_{MV}$  maximisant  $\mathcal{L}(\theta)$ , ou, de manière équivalente,  $L(\theta)$  (car le logarithme est croissant). Connaissant  $(x_1, \dots, x_N)$ ,  $\theta_{MV}$  est alors le paramètre le plus plausible parmi les paramètres possibles.

#### Exemple 5.1

On suppose  $(x_1, x_2, \dots, x_N)$  réalisations de variables aléatoires i.i.d. gaussiennes. La vraisemblance s'écrit :

$$\mathcal{L}(\mu, \sigma^2) = \frac{1}{\sigma^N (2\pi)^{N/2}} \exp\left(-\frac{1}{2} \sum_{n=1}^N \left(\frac{x_n - \mu}{\sigma}\right)^2\right)$$

et la log-vraisemblance s'écrit plus simplement :

$$L(\mu, \sigma^2) = -\log(\sigma^N (2\pi)^{N/2}) - \frac{1}{2} \sum_{n=1}^N \left(\frac{x_n - \mu}{\sigma}\right)^2$$

Les dérivées partielles de  $L$  sont alors :

$$\begin{cases} \frac{\partial L}{\partial \mu}(\mu, \sigma^2) = -\frac{1}{\sigma^2} \sum_{n=1}^N (x_n - \mu) \\ \frac{\partial L}{\partial \sigma^2}(\mu, \sigma^2) = \frac{1}{2\sigma^4} \sum_{n=1}^N (x_n - \mu)^2 - \frac{N}{2\sigma^2} \end{cases}$$

et s'annulent pour :

$$\begin{cases} \mu_{MV} = \frac{1}{N} \sum_{i=1}^N x_i \\ \sigma_{MV}^2 = \frac{1}{N} \sum_{n=1}^N (x_n - \mu_{MV})^2 \end{cases}$$

qui sont les estimateurs du maximum de vraisemblance de la moyenne et de la variance.

On remarque que l'estimateur du maximum de vraisemblance de la variance est biaisé. Néanmoins, lorsque  $N$  est assez grand, le biais devient négligeable.

1. [https://fr.wikipedia.org/wiki/Fonction\\_de\\_vraisemblance](https://fr.wikipedia.org/wiki/Fonction_de_vraisemblance)

Dans l'exemple précédent, on a pu faire les calculs et trouver une expression explicite de  $\theta_{MV}$ . Si ce n'est pas possible, on utilise une méthode d'optimisation numérique.

### 5.2.2 Une distribution très utile : le mélange de gaussiennes

Le théorème central limite justifie la modélisation de nombreux phénomènes aléatoires par la loi normale. Néanmoins, différentes sous-populations peuvent parfois être remarquées dans les observations. Ces sous-populations peuvent chacune présenter leur forme propre, et chacune d'entre elles peut éventuellement être bien modélisée par une gaussienne. La densité de l'ensemble des observations apparaît alors comme une superposition de densités gaussiennes.

Cela motive la définition du modèle de mélange de gaussiennes (*Gaussian mixture model*, ou GMM), la densité sur un espace d'observations  $\mathbb{R}^d$  étant exprimée comme :

$$\phi_{\theta}(\mathbf{x}) = \sum_{m=1}^M \pi_m \mathcal{G}_{\boldsymbol{\mu}_m, \Sigma_m}(\mathbf{x})$$

où  $\sum_{m=1}^M \pi_m = 1$  et  $\mathcal{G}_{\boldsymbol{\mu}_m, \Sigma_m}$  désigne la densité gaussienne de moyenne  $\boldsymbol{\mu}_m$  et matrice de covariance  $\Sigma_m$  :

$$\mathcal{G}_{\boldsymbol{\mu}_m, \Sigma_m}(\mathbf{x}) = \frac{1}{(2\pi)^{d/2} |\Sigma_m|^{1/2}} e^{-(\mathbf{x} - \boldsymbol{\mu}_m)^T \Sigma_m^{-1} (\mathbf{x} - \boldsymbol{\mu}_m) / 2}$$

Les paramètres d'un mélange de gaussiennes sont alors  $\theta = (\pi_m, \boldsymbol{\mu}_m, \Sigma_m)_{m \in \{1, \dots, M\}}$ .

Dans ce cas, il n'y a pas de formule explicite permettant de trouver les paramètres maximisant la vraisemblance. Néanmoins, il est possible de démontrer que l'algorithme numérique Espérance-Maximisation (EM), décrit ci-après, permet d'estimer ces paramètres à partir des observations.

La figure 5.3 montre une modélisation de densité comme un mélange de gaussiennes.

**Algorithme Espérance-Maximisation (EM)** La log-vraisemblance de  $N$  observations  $\mathbf{x}_n$  indépendantes identiquement distribuées selon un mélange de gaussiennes s'écrit :

$$L(\theta) = \sum_{n=1}^N \log \left( \sum_{m=1}^M \pi_m \mathcal{G}_{\boldsymbol{\mu}_m, \Sigma_m}(\mathbf{x}_n) \right)$$

Il va être difficile de trouver explicitement le minimum en calculant les dérivées partielles. . .

Écrivons tout de même les conditions de stationnarité sous la contrainte  $\sum_m \pi_m = 1$  : cela consiste à annuler les dérivées partielles du Lagrangien  $L(\theta) + \lambda (\sum_m \pi_m - 1)$ .

D'une part, si  $\gamma_{nm} = \frac{\pi_m \mathcal{G}_{\boldsymbol{\mu}_m, \Sigma_m}(\mathbf{x}_n)}{\sum_{l=1}^M \pi_l \mathcal{G}_{\boldsymbol{\mu}_l, \Sigma_l}(\mathbf{x}_n)}$  :

$$\frac{\partial L}{\partial \boldsymbol{\mu}_m} = \sum_{n=1}^N \gamma_{nm} \Sigma_m^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_m) = 0$$

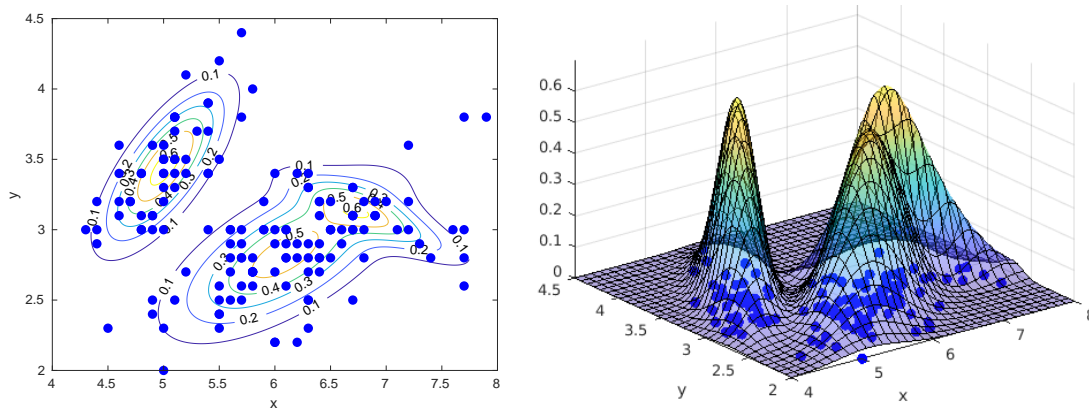


FIGURE 5.3 – La densité des observations bidimensionnelles (représentées par des points) est ici modélisée comme un mélange de trois gaussiennes. Les paramètres du mélange sont estimés par l’algorithme EM. À gauche : représentation des lignes de niveau de la densité dans le plan des observations. Rappelons que les lignes de niveau d’une densité gaussienne bidimensionnelle sont des ellipses (cf. annexe A.3.3). À droite : représentation tridimensionnelle : les observations sont représentées dans le plan  $x$ - $y$ , et l’axe des  $z$  représente la valeur de la densité estimée.

Donc en notant  $N_m = \sum_n \gamma_{nm}$  :

$$\boldsymbol{\mu}_m = \frac{1}{N_m} \sum_{n=1}^N \gamma_{nm} \mathbf{x}_n$$

D’autre part, par un raisonnement similaire :

$$\boldsymbol{\Sigma}_m = \frac{1}{N_m} \sum_{n=1}^N \gamma_{nm} (\mathbf{x}_n - \boldsymbol{\mu}_m)(\mathbf{x}_n - \boldsymbol{\mu}_m)^T$$

Enfin,

$$\begin{aligned} \frac{\partial}{\partial \pi_m} \left( L(\boldsymbol{\theta}) + \lambda \left( \sum_m \pi_m - 1 \right) \right) &= \sum_{n=1}^N \frac{\mathcal{G}_{\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m}(\mathbf{x}_n)}{\sum_{l=1}^M \pi_l \mathcal{G}_{\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l}(\mathbf{x}_n)} + \lambda \\ &= \sum_{n=1}^N \frac{\gamma_{nm}}{\pi_m} + \lambda = 0 \end{aligned}$$

et comme  $\sum_m \pi_m = 1 = \sum_m \gamma_{nm} = 1$  (par définition),  $\lambda = -N$  (car  $\lambda \pi_m = -\sum_n \gamma_{nm}$ ).

On peut conclure :

$$\pi_m = \frac{1}{N} \sum_{n=1}^N \gamma_{nm}$$

Le problème est que  $\pi_m, \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m$  dépendent de  $\gamma_{nm}$ , qui dépend lui-même de ces paramètres.

On propose donc l'algorithme itératif suivant :

Initialisation :  $\pi_m^0, \boldsymbol{\mu}_m^0, \Sigma_m^0$  connus pour tout  $1 \leq m \leq M$   
(par exemple : initialisation aléatoire).

Jusqu'à convergence de  $L(\boldsymbol{\theta}^i)$ , répéter, pour  $i \geq 0$  :

— E-step :

$$\forall m \in \{1, \dots, M\}, \forall n \in \{1, \dots, N\}, \gamma_{nm}^{i+1} = \frac{\pi_m^i \mathcal{G}_{\boldsymbol{\mu}_m^i, \Sigma_m^i}(\mathbf{x}_n)}{\sum_{l=1}^M \pi_l^i \mathcal{G}_{\boldsymbol{\mu}_l^i, \Sigma_l^i}(\mathbf{x}_n)}$$

— M-step :

$$\forall m \in \{1, \dots, M\}, \begin{cases} \boldsymbol{\mu}_m^{i+1} = \frac{1}{\sum_n \gamma_{nm}^i} \sum_{n=1}^N \gamma_{nm}^i \mathbf{x}_n \\ \Sigma_m^{i+1} = \frac{1}{\sum_n \gamma_{nm}^i} \sum_{n=1}^N \gamma_{nm}^i (\mathbf{x}_n - \boldsymbol{\mu}_m^i)(\mathbf{x}_n - \boldsymbol{\mu}_m^i)^T \\ \pi_m^{i+1} = \frac{1}{N} \sum_{n=1}^N \gamma_{nm}^i \end{cases}$$

— incrémenter  $i$

**Proposition 5.1** *Cet algorithme converge vers un maximum local ou un point-selle de la vraisemblance  $\mathcal{L}$  ou, de manière équivalente, de la log-vraisemblance  $L$ .*

L'algorithme EM fournit donc, à partir d'un ensemble d'observations, les estimateurs du maximum de vraisemblance des paramètres d'un mélange de  $M$  gaussiennes.

**Choix du nombre  $M$  de gaussiennes dans le mélange** On peut se demander comment choisir le nombre de gaussiennes devant intervenir dans la modélisation d'un ensemble d'observations.

Remarquons que plus  $M$  est grand, plus le modèle pourra s'adapter aux données et fournir une vraisemblance élevée. Le cas extrême  $M = N$  l'illustre bien : chaque observation est alors le centre d'une gaussienne dont la variance peut être prise aussi petite que l'on veut, la vraisemblance pouvant donc être très grande.

Par conséquent, il faut trouver un compromis entre vraisemblance et complexité du modèle. Différents critères ont été proposés. En particulier, on peut choisir de minimiser (et pas maximiser : il faut noter le signe devant  $L$ ) l'un des critères suivants, qui réalise un compromis entre log-vraisemblance et nombre de paramètres.

— *Bayesian's information criterion* (BIC) :

$$\text{BIC} = -2L(\boldsymbol{\theta}) + \#\{\boldsymbol{\theta}\} \log(N)$$

— *Akaike's information criterion* (AIC) :

$$\text{AIC} = -2L(\boldsymbol{\theta}) + 2\#\{\boldsymbol{\theta}\}$$



où :  $\#\{\theta\}$  désigne le nombre de paramètres du modèle. Un modèle de mélange général vérifie  $\#\{\theta\} = M - 1 + Md + Md(d + 1)/2$ . En effet, il faut estimer  $M - 1$  scalaires  $\pi_m$  sous contrainte  $\sum \pi_m = 1$ ,  $M$  vecteurs  $\mu_m$  de  $\mathbb{R}^d$ , et  $M$  matrices symétriques  $\Sigma_m$  d'ordre  $d$ . Dans un modèle où les matrices de covariances sont diagonales (pour tout  $m$ ,  $\Sigma_m = \sigma_m \text{Id}$ ),  $\#\{\theta\} = M - 1 + Md + M$ .

Plusieurs critères d'information sont possibles, que l'on peut justifier par différentes considérations que l'on ne détaillera pas.

### 5.3 Le retour de la malédiction de la dimension

L'estimation d'une densité de probabilité sur un espace de grande dimension butte généralement sur la malédiction de la dimension.

Nous avons évoqué au chapitre 2 (section 2.1.2) le problème du nombre d'hypercubes de côté  $1/n$  nécessaires pour remplir l'hypercube de côté 1 en dimension  $d$ . Les  $n^d$  hypercubes nécessaires rendent l'estimation de densité par histogrammes impraticable en grande dimension.

L'estimation de densité par fenêtres de Parzen est également limitée par la dimension : en dimension grande les fenêtres couvrent une partie de l'espace trop petite, sauf à utiliser une largeur de bande grande avec les inconvénients que l'on a soulignés. De la même manière, dans l'estimation par plus proches voisins, il faudrait que la boule contenant les  $K$  plus proches voisins soit « pas trop grosse » pour pouvoir supposer la densité constante, mais en grande dimension les observations sont toutes relativement éloignées.

Les méthodes d'estimation paramétrique nécessitent l'estimation de quelques paramètres seulement, et imposent la forme de la distribution cherchée. Par exemple, on suppose souvent les observations distribuées selon une densité gaussienne de moyenne  $\mu \in \mathbb{R}^d$  et de matrice de covariance  $\Sigma$  d'ordre  $d$ , ou selon un mélange de telles gaussiennes. Ceci facilite l'estimation de densités en grande dimension.

L'expérience suivante montre cependant que l'estimation de la matrice de covariance est également limitée par la malédiction de la dimension. On suppose disposer de  $N$  observations  $x_i$  indépendantes identiquement distribuées selon une loi normale  $\mathcal{N}_d(0, \Sigma)$ . Une estimation de  $\Sigma$  est fournie par :

$$\hat{\Sigma} = \frac{1}{N-1} \sum_{n=1}^N x_n x_n^T$$

On remarque déjà que si  $N < d$ ,  $\hat{\Sigma}$  n'est pas inversible, ce qui est tout de même problématique pour une matrice de covariance. La situation n'est pas davantage favorable lorsqu'on dispose de plus d'observations que la dimension. Définissons la norme de Frobenius normalisée par la dimension :

$$\|\Sigma - \hat{\Sigma}\|_{\text{NFn}} = \sqrt{\frac{1}{d} \sum_{i=1}^d \lambda_i^2}$$

où les  $\lambda_i$  sont les valeurs propres de la matrice symétrique  $\Sigma - \hat{\Sigma}$ . Cette norme doit converger vers 0 lorsque le nombre  $N$  d'observations augmente. Le graphique de la figure 5.4 nous montre que la convergence ralentit significativement lorsque la dimension augmente.

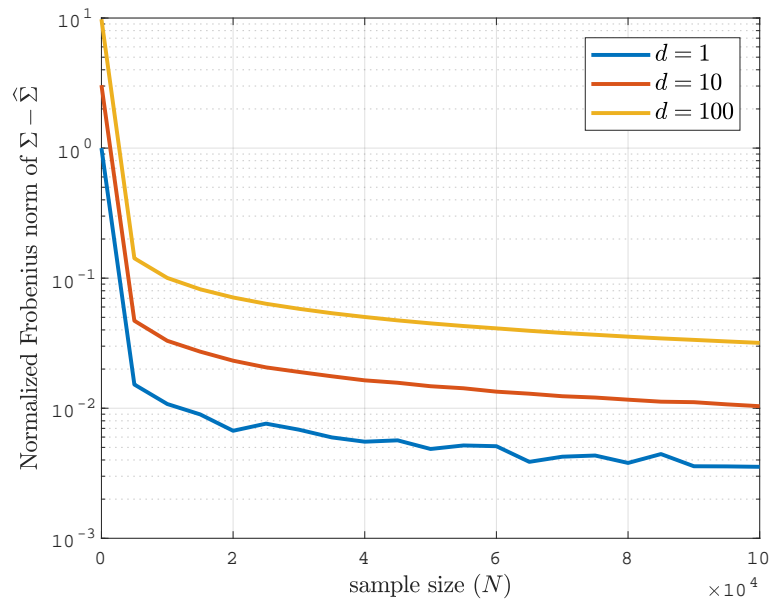


FIGURE 5.4 – Estimation de la matrice de covariance à partir de  $N$  observations générées selon une loi gaussienne. On voit que la vitesse de convergence décroît fortement avec la dimension. Par exemple, il faut de l'ordre de 10 fois plus d'observations pour atteindre la même précision en dimension 10 qu'en dimension 1, et de l'ordre de 100 fois plus en dimension 100.

Comme discuté au chapitre 2, la malédiction de la dimension nous incite en pratique à réduire le nombre de paramètres à estimer. Une possibilité est de supposer les composantes des observations statistiquement indépendantes, ce qui permet de simplifier la loi jointe sur  $\mathbb{R}^d$  en le produit des  $d$  lois marginales sur  $\mathbb{R}$ . C'est l'idée du classifieur naïf de Bayes que l'on détaillera au chapitre suivant. Dans le cas de marginales gaussiennes de paramètres  $\mu_j, \sigma_j$ , on peut écrire la densité jointe de la façon suivante :

$$\begin{aligned}
 p(\mathbf{x}) &= \prod_{j=1}^d \frac{1}{\sqrt{2\pi}\sigma_j} e^{-|x_j - \mu_j|^2 / (2\sigma_j^2)} \\
 &= \frac{1}{(2\pi)^{d/2} \prod_j \sigma_j} e^{-\sum_j |x_j - \mu_j|^2 / (2\sigma_j^2)} \\
 &= \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} e^{-(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}) / 2}
 \end{aligned}$$

où  $\boldsymbol{\mu}$  est le vecteur des  $\mu_j$ ,  $\Sigma$  est la matrice diagonale formée des variances  $\sigma_j^2$  et  $|\Sigma|$  désigne son déterminant.

Supposer l'indépendance permet de passer d'une matrice de covariance générale possédant  $d(d-1)/2$  paramètres à une matrice de covariance diagonale à  $d$  paramètres. Estimer  $\Sigma$  nécessite alors nettement moins d'observations.

## 5.4 Pour approfondir...

**L'algorithme EM vu comme un *soft k-means*** La densité d'un mélange de gaussiennes s'écrit sous la forme :

$$f(\mathbf{x}) = \sum_{m=1}^M \pi_m \mathcal{G}_{\boldsymbol{\mu}_m, \Sigma_m}(\mathbf{x})$$

On peut interpréter les  $\pi_m$  comme la probabilité a priori  $p(\mathcal{C}_m)$  d'une classe, et les  $\mathcal{G}_{\boldsymbol{\mu}_m, \Sigma_m}(\mathbf{x})$  comme les vraisemblances  $p(\mathbf{x}|\mathcal{C}_m)$  de ces classes.

Ainsi, le terme  $\gamma_{nm} = \frac{\pi_m \mathcal{G}_{\boldsymbol{\mu}_m, \Sigma_m}(\mathbf{x}_n)}{\sum_{l=1}^M \pi_l \mathcal{G}_{\boldsymbol{\mu}_l, \Sigma_l}(\mathbf{x}_n)}$  calculé dans l'algorithme EM apparaît comme la probabilité a posteriori :

$$p(\mathcal{C}_m|\mathbf{x}_n) = \frac{p(\mathcal{C}_m)p(\mathbf{x}_n|\mathcal{C}_m)}{\sum_{l=1}^M p(\mathcal{C}_l)p(\mathbf{x}_n|\mathcal{C}_l)}$$

Étant données des observations modélisées comme un mélange de gaussiennes dont on a estimé les paramètres par EM, on peut donc définir la règle de classification suivante : l'observation  $\mathbf{x}_n$  est classée dans  $\mathcal{C}_{m^*}$  telle que  $m^* = \arg \max_{1 \leq m \leq M} \gamma_{nm}$ .

Examinons le cas particulier où les gaussiennes ont toutes la même matrice de covariance diagonale  $\Sigma_m = \sigma^2 \mathbf{I}_d$ , où  $\mathbf{I}_d$  désigne la matrice identité d'ordre  $d$ . Dans ce cas,

$$\mathcal{G}_{\boldsymbol{\mu}_m, \sigma^2 \mathbf{I}_d}(\mathbf{x}) = \frac{1}{(2\pi)^{d/2} \sigma^d} e^{-\|\mathbf{x} - \boldsymbol{\mu}_m\|_2^2 / (2\sigma^2)}$$

Donc lorsque  $\sigma \rightarrow 0$ ,

$$\frac{\pi_m \mathcal{G}_{\boldsymbol{\mu}_m, \sigma^2 \mathbf{I}_d}(\mathbf{x})}{\sum_{l=1}^M \pi_l \mathcal{G}_{\boldsymbol{\mu}_l, \sigma^2 \mathbf{I}_d}(\mathbf{x})}$$

converge vers :

$$\begin{cases} 1 & \text{si } m \text{ est le maximum des } \pi_m \mathcal{G}_{\boldsymbol{\mu}_m, \sigma^2 \mathbf{I}_d}(\mathbf{x}) \\ 0 & \text{sinon} \end{cases}$$

c'est-à-dire :

$$\forall n, \lim_{\sigma \rightarrow 0} \gamma_{nm} = \begin{cases} 1 & \text{si } m = \arg \min_l \|\mathbf{x}_n - \boldsymbol{\mu}_l\|_2 \\ 0 & \text{sinon} \end{cases}$$

En effet, tous les  $\pi_l \mathcal{G}_{\boldsymbol{\mu}_l, \sigma^2 \mathbf{I}_d}$  tendent vers 0 lorsque  $\sigma \rightarrow 0$ , mais on peut conclure en écrivant :

$\frac{\pi_m \mathcal{G}_{\boldsymbol{\mu}_m, \sigma^2 \mathbf{I}_d}(\mathbf{x})}{\sum_{l=1}^M \pi_l \mathcal{G}_{\boldsymbol{\mu}_l, \sigma^2 \mathbf{I}_d}(\mathbf{x})} = \frac{1}{1 + \sum_{l \in \{1, \dots, M\} \setminus m} \alpha_l}$  avec  $\alpha_l \rightarrow 0$  lorsque  $m$  maximise  $\pi_m \mathcal{G}_{\boldsymbol{\mu}_m, \sigma^2 \mathbf{I}_d}(\mathbf{x})$  (détails laissés en exercice).

On remarque que les  $\pi_m$  et les  $\Sigma_m = \sigma^2 \mathbf{I}_d$  n'interviennent plus dans la définitions des  $\gamma_{nm}$ .

De cette manière, l'algorithme des  $k$ -moyennes (voir l'algorithme de Lloyd au chapitre 3) peut être vue comme un cas limite d'EM (ou EM comme un *soft k-means*), en écrivant à l'étape  $i$  :

— E-step :

$$\forall m, n, \gamma_{nm}^{i+1} = \begin{cases} 1 & \text{si } m = \operatorname{argmin}_l \|\mathbf{x}_n - \boldsymbol{\mu}_l\|_2 \\ 0 & \text{sinon} \end{cases}$$

$$\mathcal{C}_m^i = \{\mathbf{x}_n \text{ t.q. } \gamma_{nm}^{i+1} = 1\}$$

— M-step :

$$\forall m, n, \boldsymbol{\mu}_m^{i+1} = \frac{1}{\#\mathcal{C}_m^i} \sum_{\mathbf{x} \in \mathcal{C}_m^i} \mathbf{x}$$

L'algorithme « mélange de gaussiennes + EM » permet de dépasser les limites de l'algorithme de Lloyd pour les  $k$ -moyennes : les groupes ne sont pas forcément isotropes et homogènes (par prise en compte de la matrice de covariance), et les appartenances à un groupe sont données par les probabilités  $\gamma_{nm}$  plutôt que par une affectation univoque. On peut fixer le nombre de groupes à l'aide des critères d'information AIC ou BIC. L'inconvénient d'EM est la difficulté d'estimer en pratique les matrices de covariance dès que la dimension augmente. La convergence est généralement plus lente que pour l'algorithme des  $k$ -moyennes.

# Mise en œuvre du classifieur de Bayes : classifieur naïf, régression logistique, plus proches voisins

Dans ce chapitre, nous nous intéressons au problème de la classification supervisée et à la manière de mettre en œuvre le classifieur de Bayes. En particulier, nous allons voir le classifieur naïf de Bayes, la régression logistique, et l'algorithme des plus proches voisins.

Nous avons établi au chapitre 4 que le classifieur de Bayes est optimal sous hypothèse d'un coût d'erreur de classification 0-1. Ce classifieur consiste à associer une observation  $\mathbf{x} \in \mathbb{R}^d$  à la classe  $\mathcal{C}_k$  qui maximise, parmi les  $K$  classes  $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_K$ , la probabilité a posteriori  $p(\mathcal{C}_k|\mathbf{x})$ , ce qui revient à maximiser le produit  $p(\mathcal{C}_k)p(\mathbf{x}|\mathcal{C}_k)$  de la probabilité a priori et de la vraisemblance. Cela nécessite de connaître soit les probabilités a posteriori  $p(\mathcal{C}_k|\mathbf{x})$ , soit les probabilité a priori  $p(\mathcal{C}_k)$  et les vraisemblances  $p(\mathbf{x}|\mathcal{C}_k)$ . Le chapitre 5 présente quelques méthodes d'estimation de probabilités, mais souligne également que l'estimation des vraisemblances est sujette à la malédiction de la dimension : en pratique on ne peut pas estimer  $p(\mathbf{x}|\mathcal{C}_k)$  de manière fiable dès que la dimension de l'espace des observations est relativement grande, à moins de disposer de gigantesques bases d'apprentissage.

Différentes hypothèses vont nous permettre de simplifier l'estimation pratique des probabilités afin de mettre en œuvre des « approximations » du classifieur idéal de Bayes.

## 6.1 Classifieur naïf de Bayes

### 6.1.1 Indépendance conditionnelle

Une manière de lutter contre la malédiction de la dimension est de supposer les composantes de  $\mathbf{x} = (x^1, x^2, \dots, x^d) \in \mathbb{R}^d$  indépendantes conditionnellement à chaque classe  $\mathcal{C}_k$ . La loi jointe étant alors le produit des lois marginales, on peut écrire l'égalité suivante pour

chaque classe  $\mathcal{C}_k$  :

$$p(\mathbf{x}|\mathcal{C}_k) = \prod_{i=1}^d p(x^i|\mathcal{C}_k)$$

Le gros avantage de cette approche est qu'on est ramené à l'estimation des  $d$  lois de probabilité  $p(x^i|\mathcal{C}_k)$  sur  $\mathbb{R}$  à la place de la loi  $p(\mathbf{x}|\mathcal{C}_k)$  sur  $\mathbb{R}^d$ .

On définit ainsi le classifieur naïf de Bayes :

$$f(\mathbf{x}) = \underset{\mathcal{C}_k}{\operatorname{argmax}} p(\mathcal{C}_k) \prod_{i=1}^d p(x^i|\mathcal{C}_k)$$

Ce classifieur est qualifié de naïf car il fait l'hypothèse que les composantes sont conditionnellement indépendantes, ce qui est en pratique très optimiste. Ce classifieur présente malgré tout de bonnes performances dans un certain nombre de cas pratiques.

### 6.1.2 Classifieur naïf de Bayes gaussien

Souvent, on suppose de plus les  $K$  distributions  $p(x^i|\mathcal{C}_k)$  gaussiennes. Elles sont donc caractérisées par leur moyennes et écarts-types  $\mu_{i,k}, \sigma_{i,k}$ , à estimer à partir des données, et s'écrivent :

$$p(x^i|\mathcal{C}_k) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{1}{2}((x^i-\mu_{i,k})^2/\sigma_i^2)}$$

On obtient alors le classifieur naïf de Bayes gaussien (*Gaussian naive Bayes classifier*, GNB). Comme discuté au paragraphe 5.3 du chapitre 5, estimer ces paramètres réels nécessite bien moins d'observations que pour estimer les paramètres d'une loi gaussienne sur  $\mathbb{R}^d$ .

La notion d'indépendance conditionnellement à la classe est illustrée par la figure 6.1 : les composantes des observations d'étiquette  $\mathcal{C}_k$  sont non-corrélées entre elles, mais les composantes peuvent présenter une corrélation dans leur ensemble.

Dans le cas biclasse, le classifieur naïf de Bayes est en fait un classifieur linéaire pour un certain nombre de choix de distributions de probabilités  $p(x^i|\mathcal{C}_k)$ . En particulier, c'est le cas dans le cas gaussien si de plus, pour toute composante  $i$ ,  $\sigma_{i,1} = \sigma_{i,2}$ , noté ci-dessous  $\sigma_i$ .

En effet, sous cette hypothèse on doit calculer

$$p(\mathcal{C}_k|\mathbf{x}) = p(\mathcal{C}_k) \prod_{i=1}^d \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{1}{2}((x^i-\mu_{i,k})^2/\sigma_i^2)}$$

pour  $k \in \{1, 2\}$ .

La règle de décision est de choisir la classe  $\mathcal{C}_1$  si  $p(\mathcal{C}_1|\mathbf{x}) > p(\mathcal{C}_2|\mathbf{x})$  ou, de manière équivalente, si  $\log(p(\mathcal{C}_1|\mathbf{x})/p(\mathcal{C}_2|\mathbf{x})) > 0$ . La frontière de classification vérifie alors (à développer en exercice) :

$$\log\left(\frac{p(\mathcal{C}_1|\mathbf{x})}{p(\mathcal{C}_2|\mathbf{x})}\right) = \alpha_0 + \boldsymbol{\alpha}_1 \cdot \mathbf{x}$$

où

$$\alpha_0 = \log\left(\frac{p(\mathcal{C}_1)}{p(\mathcal{C}_2)}\right) + \sum_{i=1}^d \frac{\mu_{i,2}^2 - \mu_{i,1}^2}{2\sigma_i^2}$$

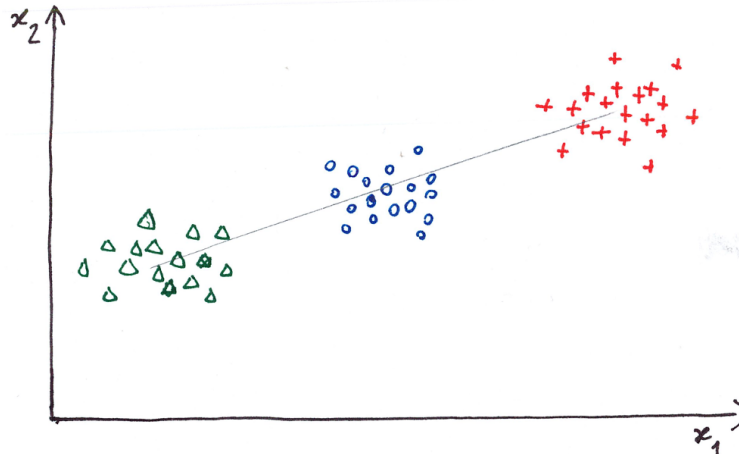


FIGURE 6.1 – Illustration de l'indépendance conditionnelle à la classe. Les observations  $(x_1, x_2)$  sont générées aléatoirement selon une loi telle que pour chaque classe  $\mathcal{C}_k$ ,  $p(x_1, x_2 | \mathcal{C}_k) = p(x_1 | \mathcal{C}_k)p(x_2 | \mathcal{C}_k)$  mais  $p(x_1, x_2) \neq p(x_1)p(x_2)$ . On constate que les observations de chaque classe ont des composantes non corrélées entre elles (chaque classe présente un aspect « non structuré »), mais aussi que les observations dans leur ensemble ont des composantes corrélées (d'où la tendance marquée par la droite superposée).

$$\alpha_1 = \left( \frac{\mu_{i,1} - \mu_{i,2}}{\sigma_i^2} \right)_{1 \leq i \leq d}$$

Nous avons donc démontré, sous l'hypothèse d'égalité des variances des composantes ( $\forall i, \sigma_{i,1} = \sigma_{i,2}$ ), que le classifieur naïf de Bayes gaussien sépare linéairement les deux classes. Autrement dit, la frontière de classification est un hyperplan de  $\mathbb{R}^d$ . Attention, ceci n'est pas vrai si les variances ne sont pas supposées égales. Dans ce cas, le calcul précédent montrerait que la fonction de décision est quadratique, conduisant en dimension  $d = 2$  à une frontière de décision en ellipse, parabole, ou hyperbole (voir section 6.5).

## 6.2 Régression logistique

Nous nous restreignons au cas d'un problème de classification à deux classes  $\mathcal{C}_1$  et  $\mathcal{C}_2$ .

Le théorème de Bayes nous permet d'écrire la probabilité a posteriori  $p(\mathcal{C}_1 | \mathbf{x})$  de la manière suivante :

$$\begin{aligned} p(\mathcal{C}_1 | \mathbf{x}) &= \frac{p(\mathcal{C}_1)p(\mathbf{x} | \mathcal{C}_1)}{p(\mathcal{C}_1)p(\mathbf{x} | \mathcal{C}_1) + p(\mathcal{C}_2)p(\mathbf{x} | \mathcal{C}_2)} \\ &= \frac{1}{1 + \frac{p(\mathcal{C}_2)p(\mathbf{x} | \mathcal{C}_2)}{p(\mathcal{C}_1)p(\mathbf{x} | \mathcal{C}_1)}} \end{aligned}$$

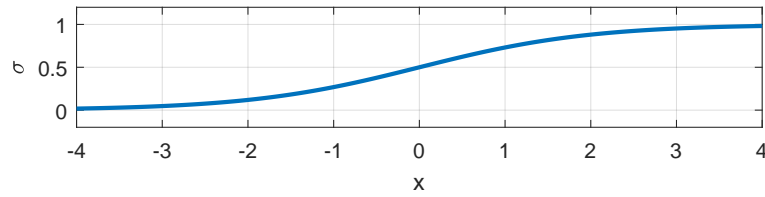


FIGURE 6.2 – Graphique de la fonction logistique (ou sigmoïde).

Posons :

$$f(\mathbf{x}) = \log\left(\frac{p(\mathbf{x}|\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_2)}\right) + \log\left(\frac{p(\mathcal{C}_1)}{p(\mathcal{C}_2)}\right)$$

Ainsi :

$$p(\mathcal{C}_1|\mathbf{x}) = \frac{e^{f(\mathbf{x})}}{1 + e^{f(\mathbf{x})}} = \frac{1}{1 + e^{-f(\mathbf{x})}}$$

La fonction définie sur  $\mathbb{R}$  par  $\sigma(t) = \frac{1}{1+e^{-t}}$  est appelée fonction logistique (ou sigmoïde), et est représentée par la figure 6.2.

On a déjà vu qu’une manière de contrer la malédiction de la dimension est de faire des hypothèses restrictives sur les distributions de probabilité à estimer. Ici, nous allons supposer que  $f$  est une fonction affine de  $\mathbf{x}$  :  $f(\mathbf{x}) = \beta_0 + \boldsymbol{\beta}_1 \cdot \mathbf{x}$ , où  $\beta_0 \in \mathbb{R}$  et  $\boldsymbol{\beta}_1 \in \mathbb{R}^d$ .

Si on sait estimer  $\beta_0$  et  $\boldsymbol{\beta}_1$  à partir des observations, le classifieur de Bayes fournit la règle de décision suivante :

$$\begin{cases} \mathbf{x} \text{ classé dans } \mathcal{C}_1 & \text{si } f(\mathbf{x}) > 0 \quad (\text{c'est-à-dire } p(\mathcal{C}_1|\mathbf{x}) = \sigma(f(\mathbf{x})) > 1/2) \\ \mathbf{x} \text{ classé dans } \mathcal{C}_2 & \text{si } f(\mathbf{x}) < 0 \quad (\text{c'est-à-dire } p(\mathcal{C}_2|\mathbf{x}) = 1 - p(\mathcal{C}_1|\mathbf{x}) = \sigma(-f(\mathbf{x})) > 1/2) \end{cases}$$

La fonction  $f$  étant affine, les deux classes sont donc séparées par un hyperplan de  $\mathbb{R}^d$ .

Au passage, lorsque  $p(\mathbf{x}|\mathcal{C}_1)$  et  $p(\mathbf{x}|\mathcal{C}_2)$  sont des distributions gaussiennes de même matrice de covariance  $\Sigma$ , la fonction  $f$  est bien affine, ceci n’est pas une hypothèse simplificatrice. En effet, si  $p(\mathbf{x}|\mathcal{C}_k) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)}$  pour  $k = 1$  ou  $k = 2$ , alors :

$$\begin{aligned} f(\mathbf{x}) &= -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_1)^T \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_1) + \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_2)^T \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_2) + \log\left(\frac{p(\mathcal{C}_1)}{p(\mathcal{C}_2)}\right) \\ &= -\frac{1}{2}\boldsymbol{\mu}_1^T \Sigma^{-1} \boldsymbol{\mu}_1 + \mathbf{x}^T \Sigma^{-1} \boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_2^T \Sigma^{-1} \boldsymbol{\mu}_2 - \mathbf{x}^T \Sigma^{-1} \boldsymbol{\mu}_2 + \log\left(\frac{p(\mathcal{C}_1)}{p(\mathcal{C}_2)}\right) \end{aligned}$$

d’où  $f(\mathbf{x}) = \beta_0 + \boldsymbol{\beta}_1 \cdot \mathbf{x}$  avec :  $\beta_0 = \frac{1}{2}\boldsymbol{\mu}_2^T \Sigma^{-1} \boldsymbol{\mu}_2 - \frac{1}{2}\boldsymbol{\mu}_1^T \Sigma^{-1} \boldsymbol{\mu}_1 + \log\left(\frac{p(\mathcal{C}_1)}{p(\mathcal{C}_2)}\right)$  et  $\boldsymbol{\beta}_1 = \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$

### 6.2.1 Estimation des paramètres : régression logistique

Nous nous plaçons sous l’hypothèse simplificatrice  $f(\mathbf{x}) = \beta_0 + \boldsymbol{\beta}_1 \cdot \mathbf{x}$ .



On estime alors les paramètres  $(\beta_0, \boldsymbol{\beta}_1)$  par maximisation de la vraisemblance conditionnelle (voir section 5.2.1 au chapitre 5) : on cherche les paramètres qui maximisent la vraisemblance des étiquettes conditionnées aux observations. Les étiquettes  $y$  sont distribuées selon  $p(y|\mathbf{x})$ . En notant  $y = 1$  si  $\mathbf{x}$  est dans  $\mathcal{C}_1$ , et  $y = 0$  si  $\mathbf{x}$  est dans  $\mathcal{C}_2$ , on a :  $p(y|\mathbf{x}) = p(\mathcal{C}_1|\mathbf{x})^y p(\mathcal{C}_2|\mathbf{x})^{1-y}$  (on vérifie bien que si  $y = 1$ ,  $p(y|\mathbf{x}) = p(\mathcal{C}_1|\mathbf{x})$ , et si  $y = 0$ ,  $p(y|\mathbf{x}) = p(\mathcal{C}_2|\mathbf{x}) = 1 - p(\mathcal{C}_1|\mathbf{x})$ ).

La log-vraisemblance conditionnelle s'exprime donc à partir d'un ensemble d'observations étiquetées  $(\mathbf{x}_n, y_n)_{1 \leq n \leq N}$  comme :

$$\begin{aligned} L(\beta_0, \boldsymbol{\beta}_1) &= \log \left( \prod_{n=1}^N (p(\mathcal{C}_1|\mathbf{x}_n))^{y_n} (1 - p(\mathcal{C}_1|\mathbf{x}_n))^{1-y_n} \right) \\ &= \sum_{n=1}^N y_n \log(p(\mathcal{C}_1|\mathbf{x}_n)) + (1 - y_n) \log(1 - p(\mathcal{C}_1|\mathbf{x}_n)) \\ &= \sum_{n=1}^N \left( y_n (\beta_0 + \boldsymbol{\beta}_1 \cdot \mathbf{x}_n) - \log(1 + \exp(\beta_0 + \boldsymbol{\beta}_1 \cdot \mathbf{x}_n)) \right) \end{aligned}$$

car  $p(\mathcal{C}_1|\mathbf{x}) = \sigma(f(\mathbf{x})) = e^{f(\mathbf{x})} / (1 + e^{f(\mathbf{x})})$  donc  $\log(p(\mathcal{C}_1|\mathbf{x})) = f(\mathbf{x}) - \log(1 + e^{f(\mathbf{x})})$  et  $\log(1 - p(\mathcal{C}_1|\mathbf{x})) = \log(1 - \sigma(f(\mathbf{x}))) = -\log(1 + e^{f(\mathbf{x})})$ .

La fonction  $L(\beta_0, \boldsymbol{\beta}_1)$  est concave en  $(\beta_0, \boldsymbol{\beta}_1)$  comme somme de fonctions concave : d'une part les fonctions affines ou nulles  $(\beta_0, \boldsymbol{\beta}_1) \mapsto y_n(\beta_0 + \boldsymbol{\beta}_1 \cdot \mathbf{x}_n)$  sont concaves et d'autre part les  $(\beta_0, \boldsymbol{\beta}_1) \mapsto -\log(1 + \exp(\beta_0 + \boldsymbol{\beta}_1 \cdot \mathbf{x}_n))$  aussi, pour toute valeur de  $n$ . Le maximum de la vraisemblance conditionnelle est donc unique (voir annexe B.1). Il est donc légitime de parler du classifieur de « la » régression logistique : quelle que soit la manière de maximiser la vraisemblance conditionnelle, on aboutira toujours aux mêmes paramètres  $(\beta_0, \boldsymbol{\beta}_1)$  (aux imprécisions numériques près). Généralement, c'est un algorithme de montée de gradient qui est utilisé pour maximiser  $L(\beta_0, \boldsymbol{\beta}_1)$  (voir annexe B.3).

Attention au vocabulaire : la régression logistique permet de déterminer  $\beta_0$  et  $\boldsymbol{\beta}_1$  ; le classifieur de la régression logistique, discuté dans le paragraphe suivant, implémente le principe du maximum a posteriori sous l'hypothèse  $p(\mathcal{C}_1|\mathbf{x}) = \sigma(\beta_0 + \boldsymbol{\beta}_1 \cdot \mathbf{x})$ .

### 6.2.2 Classifieur de la régression logistique

Le classifieur de la régression logistique est ce qu'on appelle un « classifieur linéaire », car la surface de séparation est l'hyperplan de l'espace des observations d'équation  $f(\mathbf{x}) = \beta_0 + \boldsymbol{\beta}_1 \cdot \mathbf{x} = 0$ . Rappelons que la règle de décision est (voir la section précédente) :

$$\begin{cases} \mathbf{x} \text{ classé dans } \mathcal{C}_1 & \text{si } f(\mathbf{x}) > 0 \\ \mathbf{x} \text{ classé dans } \mathcal{C}_2 & \text{si } f(\mathbf{x}) < 0 \end{cases}$$

On a montré que la log-vraisemblance conditionnelle s'écrit sous la forme :

$$L(\beta_0, \boldsymbol{\beta}_1) = \sum_{i=1}^n y_i \log(\sigma(f(\mathbf{x}_i))) + (1 - y_i) \log(\sigma(-f(\mathbf{x}_i)))$$

Lorsqu'on maximise cette fonction, on se rend compte qu'on a tout intérêt à avoir  $f(\mathbf{x}_i) = \beta_0 + \boldsymbol{\beta}_1 \cdot \mathbf{x}_i > 0$  lorsque  $y_i = 1$  et  $f(\mathbf{x}_i) < 0$  lorsque  $y_i = 0$ , c'est-à-dire, à bien classer les observations.

Par ailleurs, la sigmoïde est peu sensible à l'éloignement de  $\mathbf{x}_i$  au plan d'équation  $f(\mathbf{x}) = 0$ , car cette fonction admet deux asymptotes (1 en  $+\infty$  et 0 en  $-\infty$ ) : les points tels que  $|f(\mathbf{x})|$  est « grand » ont une influence dans l'expression de la log-vraisemblance limitée par cette propriété. Ce n'est pas le cas dans la régression linéaire : dans ce cas on minimise la norme au carré d'un résidu, et les observations de résidu élevé vont avoir une forte influence sur l'estimation des paramètres de la droite cherchée.

Il est possible d'introduire une régularisation dans la régression logistique en contraignant les valeurs pouvant être prises par les paramètres par l'intermédiaire de la norme euclidienne de  $\boldsymbol{\beta}_1$ . Pour ce faire, il suffit de chercher à minimiser

$$-L(\beta_0, \boldsymbol{\beta}_1) + \lambda \|\boldsymbol{\beta}_1\|_2^2$$

pour un hyperparamètre  $\lambda \geq 0$ , plutôt que maximiser la seule log-vraisemblance  $L(\beta_0, \boldsymbol{\beta}_1)$ . Au passage, remarquons que la fonction objectif est convexe, comme somme de deux fonctions convexes. La solution reste donc unique.

### 6.3 Classification au sens des $P$ plus proches voisins

La classification aux  $P$  plus proches voisins est sans doute le plus simple algorithme de classification. On dispose d'un jeu de données fait de  $N$  observations  $\mathbf{x}_1, \dots, \mathbf{x}_N$ , chaque observation faisant partie d'une des  $K$  classes  $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_K$ . Lorsqu'on veut déterminer la classe d'une nouvelle observation  $\mathbf{x}$ , on cherche alors ses  $P$  plus proches voisins parmi les  $N$  observations (au sens d'une métrique prédéfinie), et on classe  $\mathbf{x}$  dans la classe majoritaire parmi les classes des  $P$  plus proches voisins. Il se trouve que ce classifieur présente souvent de bonnes performances.

Nous allons voir que cet algorithme de classification met en œuvre le classifieur de Bayes sous certaines hypothèses (très) simplificatrices.

#### 6.3.1 Rappel : estimation aux plus proches voisins d'une distribution

On a vu au chapitre 5 (section 5.1.3) qu'il est possible d'estimer une distribution de probabilité  $\phi$  en considérant les  $P$  plus proches observations de  $\mathbf{x}$  parmi les  $N$  observations disponibles, comme :

$$\phi(\mathbf{x}) \simeq \frac{P}{N V_P(\mathbf{x})}$$

où  $V_P(\mathbf{x})$  est le volume d'une boule centrée en  $\mathbf{x}$  contenant ces  $P$  observations.

#### 6.3.2 Classification au sens du maximum a posteriori

Appliquons cette méthode à l'estimation des vraisemblances  $p(\mathbf{x}|\mathcal{C}_1), p(\mathbf{x}|\mathcal{C}_2), \dots, p(\mathbf{x}|\mathcal{C}_K)$ . Supposons que parmi les  $P$  plus proches voisins d'une observation  $\mathbf{x}$ ,  $P_1$  voisins appartiennent à la classe  $\mathcal{C}_1$ ,  $P_2$  à la classe  $\mathcal{C}_2, \dots$ , et  $P_K$  à la classe  $\mathcal{C}_K$ , de sorte que  $\sum_{k=1}^K P_k = P$ . De même,

notons  $N_1$  le nombre d'observations du jeu de données dans  $\mathcal{C}_1$ ,  $N_2$  le nombre dans  $\mathcal{C}_2, \dots$ , et  $N_K$  dans  $\mathcal{C}_K$ , de sorte que  $\sum_{k=1}^K N_k = N$ . Dans ce cas, pour tout  $k \in \{1, \dots, K\}$ ,

$$p(\mathbf{x}|\mathcal{C}_k) = \frac{P_k}{N_k V_P(\mathbf{x})}$$

Les probabilités a priori peuvent pour leur part être estimées comme la proportion d'observations du jeu de données appartenant à une certaine classe :

$$p(\mathcal{C}_k) = \frac{N_k}{N}$$

Les probabilités a posteriori sont donc, d'après le théorème de Bayes :

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathcal{C}_k)p(\mathbf{x}|\mathcal{C}_k)}{p(\mathbf{x})} = \frac{P_k}{N V_P(\mathbf{x}) p(\mathbf{x})}$$

On remarque que seul  $P_k$  dépend de  $k$ . La règle de Bayes consiste donc à classer  $\mathbf{x}$  dans la classe  $\mathcal{C}_k$  telle que  $P_k$  est maximum, c'est-à-dire dans la classe majoritaire parmi les  $P$  plus proches voisins de  $\mathbf{x}$ .

### 6.3.3 Discussion

Si cet algorithme est très simple, il présente les mêmes limites que celles discutées pour l'estimation des distributions de probabilité : problème de la dimension, « densité » des observations qui doit être suffisante autour de  $\mathbf{x} \dots$

On a intérêt à utiliser la classification aux  $P$  plus proches voisins avec  $P$  « un peu grand » ( $P = 5$ ?  $P = 10$ ?) de manière à ne pas dépendre uniquement « du » plus proche voisin et limiter ainsi le sur-apprentissage (en d'autres termes, à augmenter le biais et diminuer la fluctuation ou la variance, cf section 2.3 au chapitre 2). Néanmoins, plus  $P$  est grand, plus  $V_P$  devient grand, donc moins on a de chance de respecter l'hypothèse «  $\phi$  constante sur  $V_P(\mathbf{x})$  » qui est à la base de l'estimation des densités. Augmenter  $P$  conduit donc à s'écarter du classifieur idéal de Bayes.

Nous n'avons pas discuté des aspects algorithmiques du problème. La classification aux  $P$  plus proches voisins nécessite de trouver, comme son nom l'indique, les  $P$  plus proches voisins d'une nouvelle observation à classifier. Lorsque le jeu de données est grand ( $N = 1$  million d'observations étiquetées?), il est très coûteux de tester exhaustivement les  $N$  observations pour déterminer les  $P$  plus proches de  $\mathbf{x}$ . Une manière efficace de faire est d'organiser les observations dans un arbre  $k$ -d ( $k$ -d tree). Cependant, lorsque la dimension augmente, il se trouve que la performance d'une recherche par arbre  $k$ -d tree se rapproche de celle de la méthode exhaustive : il s'agit encore d'une manifestation de la malédiction de la dimension. Des algorithmes ont récemment été proposés pour chercher des plus proches voisins approchés (on s'autorise des erreurs dans la liste des  $P$  plus proches voisins), ou de manière adaptée à la recherche dans un espace de grande dimension, voir le paragraphe 6.5.

## 6.4 Résumé : mise en œuvre du classifieur de Bayes

Nous avons vu que le classifieur optimal pour un coût d'erreur 0-1 est le classifieur de Bayes (ou classifieur du maximum a posteriori) qui consiste à choisir la classe  $\mathcal{C}_k$  qui maximise  $p(\mathcal{C}_k|\mathbf{x})$  ou, de manière équivalente  $p(\mathcal{C}_k)p(\mathbf{x}|\mathcal{C}_k)$ . Il s'agit d'un classifieur théorique idéal car en pratique nous ne connaissons pas la distribution de probabilité des observations étiquetées.

On peut chercher à estimer les vraisemblances  $p(\mathbf{x}|\mathcal{C}_k)$  et probabilités a priori  $p(\mathcal{C}_k)$  par différentes méthodes de manière à mettre en œuvre ce classifieur. Lorsque les probabilités a priori sont considérées toutes égales, le classifieur de Bayes revient au classifieur du maximum de vraisemblance : on choisit la classe  $\mathcal{C}_k$  qui maximise  $p(\mathbf{x}|\mathcal{C}_k)$ .

Néanmoins, estimer les vraisemblances est illusoire lorsque les observations  $\mathbf{x}$  appartiennent à un espace vectoriel de grande dimension, à cause de la malédiction de la dimension. Dans ce cas, on peut supposer les composantes des observations indépendantes conditionnellement à la classe  $\mathcal{C}_k$ . C'est l'hypothèse du classifieur naïf de Bayes qui a l'avantage de nécessiter uniquement l'estimation de distributions de probabilité sur  $\mathbb{R}$ . Lorsque ces distributions sont de plus supposées gaussiennes, on parle de classifieur naïf de Bayes gaussien.

Une autre manière de contrer la malédiction de la dimension est de chercher à estimer directement les probabilités a posteriori  $p(\mathcal{C}_k|\mathbf{x})$ , sous l'hypothèse simplificatrice  $p(\mathcal{C}_k|\mathbf{x}) = \sigma(f(\mathbf{x}))$  où  $\sigma$  est la fonction logistique et  $f(\mathbf{x})$  une fonction affine de  $\mathbf{x}$ . Il s'agit de la régression logistique, qui fournit un classifieur dont la frontière de décision est un hyperplan. Bien entendu, si les deux classes ne sont pas linéairement séparables, la régression logistique souffrira de sous-apprentissage.

Nous avons vu que le classifieur des  $P$  plus proches voisins était aussi une manière de mettre en œuvre le classifieur de Bayes, sous des hypothèses assez grossières mais avec un certain succès dans bon nombre d'applications.

Dans les chapitres 8 et 9, nous allons voir deux nouveaux séparateurs linéaires, la machine à vecteur support linéaire et le perceptron de Rosenblatt. Dans les deux cas, il est possible de les étendre aux classes non linéairement séparables, ce qui fait des machines à vecteurs supports (SVM) et des réseaux de neurones artificiels les grands succès du vingt-et-unième siècle.

## 6.5 Pour approfondir...

**Modèles génératifs et modèles discriminants** La littérature distingue ces deux modèles de classification, qu'y a-t-il derrière ce vocabulaire ?

Le classifieur naïf de Bayes (en particulier dans le cas gaussien) fait partie des modèles génératifs. Ces modèles s'appuient sur les vraisemblances  $p(\mathbf{x}|\mathcal{C}_k)$  et les probabilités a priori  $p(\mathcal{C}_k)$  ; il serait possible de générer des données aléatoirement à partir de ces distributions de probabilité. Pour ce faire, on tire au hasard une classe  $l$  selon la distribution des  $p(\mathcal{C}_k)$  ( $1 \leq k \leq K$ ), puis on tire au hasard une observation selon la distribution  $p(\mathbf{x}|\mathcal{C}_l)$ .

La régression logistique fait partie des modèles discriminants, dans lesquels on modélise directement les probabilités a posteriori  $p(\mathcal{C}_k|\mathbf{x})$  qui vont servir à discriminer les classes

selon le principe du maximum a posteriori.

**Régression logistique et classification naïve de Bayes gaussienne** Dans le cas de variances des composantes  $x^i$  identiques pour toutes les classes  $\mathcal{C}_k$ , on a vu que le classifieur naïf de Bayes gaussien fournissait un séparateur linéaire, tout comme le classifieur de la régression logistique. Cependant, ils ne fournissent pas les mêmes frontières. En effet, l'estimation des paramètres ne se fait pas de la même façon dans les deux cas : on utilise des estimateurs classiques de la moyenne et de la variance des composantes dans le premier cas, et on maximise la vraisemblance conditionnelle dans le second cas.

Des arguments théoriques et expérimentaux permettent de dire que la régression logistique a de meilleures performances en généralisation que la classification naïve de Bayes gaussienne lorsque la base d'apprentissage est grande (sinon la conclusion est inversée). On pourra consulter :

A. Ng and M. Jordan, *On discriminative vs. generative classifiers : A comparison of logistic regression and naive Bayes*, Proceedings of the International Conference on Neural Information Processing Systems (NIPS), 2001.

**Au delà des classifieurs linéaires** On peut se demander pourquoi chercher des surfaces séparatrices sous forme d'hyperplan plutôt que sous forme d'une surface quadratique, ou d'une autre nature géométrique. Une réponse sera donnée dans le chapitre suivant : lorsque les observations appartiennent à un espace de grande dimension, il sera « facile » de séparer les classes par un simple hyperplan. Un classifieur linéaire étant défini par un produit scalaire, on verra également qu'il est possible de le rendre non-linéaire par l'astuce du noyau.

Il y a toutefois des situations où il est utile d'imposer la géométrie de la surface de séparation. Il est possible de choisir une surface a priori : on verra par exemple que le choix d'un noyau judicieux permet de s'assurer que les surfaces sont quadratiques. En général, on n'impose pas la surface en elle-même (comment choisir sa nature géométrique a priori?) mais on fait plutôt des hypothèses dans le cadre d'un modèle probabiliste des observations. Nous ne développerons pas ce point de vue dans le cours, mais nous allons discuter rapidement un exemple.

Plaçons-nous dans le cadre d'un problème de classification biclasse pour des observations de  $\mathbb{R}^d$ , dans lequel on modélise chaque vraisemblance  $p(\mathbf{x}|\mathcal{C}_k)$  ( $k \in \{1, 2\}$ ) par une loi gaussienne de moyenne  $\boldsymbol{\mu}_k$  et matrice de covariance  $\Sigma_k$ . Le principe du maximum a posteriori associe une observation  $\mathbf{x}$  à la classe  $\mathcal{C}_1$  si :

$$p(\mathcal{C}_1) \frac{1}{\sqrt{(2\pi)^d |\Sigma_1|}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_1)^T \Sigma_1^{-1}(\mathbf{x}-\boldsymbol{\mu}_1)} > p(\mathcal{C}_2) \frac{1}{\sqrt{(2\pi)^d |\Sigma_2|}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_2)^T \Sigma_2^{-1}(\mathbf{x}-\boldsymbol{\mu}_2)}$$

soit, en passant au logarithme :

$$K_1 - (\mathbf{x} - \boldsymbol{\mu}_1)^T \Sigma_1^{-1} (\mathbf{x} - \boldsymbol{\mu}_1) > K_2 - (\mathbf{x} - \boldsymbol{\mu}_2)^T \Sigma_2^{-1} (\mathbf{x} - \boldsymbol{\mu}_2)$$

où  $K_1$  et  $K_2$  sont deux constantes.

En développant, on se rend compte que la surface de séparation entre les deux classes est une (hyper)surface quadratique. Par exemple, si  $d = 2$  on obtient des ellipses, paraboles,

hyperboles, et des cas dégénérés comme les droites, voir annexe A.3 (on sait déjà que le maximum a posteriori conduit à un classifieur linéaire pour des matrices de covariances bien choisies, voir Section 6.2). D'autres modèles probabilistes induiraient des séparations de géométrie encore différente.

**Recherche efficace des plus proches voisins** Concernant la recherche des plus proches voisins, de manière approchée ou en grande dimension, l'article suivant :

M. Muja and D. Lowe, *Scalable Nearest Neighbor Algorithms for High Dimensional Data*, IEEE Transactions on Image Processing, vol. 36, no. 11, p. 2227-2240, 2011

ainsi que la très utile bibliothèque FLANN (*Fast Library for Approximate Nearest Neighbors*) :

<https://github.com/flann-lib/flann>

font référence.

# Méthodes ensemblistes : *bagging* et *boosting*

Ce chapitre traite de problèmes de classification supervisée. Nous proposons de combiner la réponse de plusieurs classifieurs dont les performances individuelles sont limitées, dans le but d'obtenir un « super-classifieur » plus performant. Les techniques relatives à ce problème sont appelées « méthodes ensemblistes » (*ensemble methods*).

### 7.1 Méthodes ensemblistes, classifieurs faibles et forts

Considérons un problème de classification biclasse, pour lequel on dispose de  $N$  classifieurs donnant des prédictions indépendantes. On suppose que chaque classifieur a une probabilité d'erreur de  $p < 1/2$ . Lorsque la probabilité  $p$  est juste inférieure à  $1/2$ , le classifieur est à peine meilleur qu'une réponse aléatoire. On parle alors de « classifieur faible » (*weak classifier*). Il semble naturel, connaissant la prédiction de chacun des  $N$  classifieurs pour une observation donnée, d'associer cette observation à la classe majoritaire parmi les classes prédites. Cette méthode améliore-t-elle la prédiction?

Un simple dénombrement sous hypothèse d'indépendance des prédictions nous dit que la probabilité qu'exactement  $k$  classifieurs parmi les  $N$  prédisent la mauvaise classe est (selon la loi binomiale) :

$$\binom{N}{k} p^k (1-p)^{N-k}$$

La probabilité que la majorité des classifieurs prédise la mauvaise classe est donc :

$$\sum_{k=\lceil N/2 \rceil}^N \binom{N}{k} p^k (1-p)^{N-k}$$

où  $\lceil N/2 \rceil$  désigne le plus petit entier supérieur ou égal à  $N/2$ .

Il se trouve que cette valeur décroît assez vite avec  $N$ . Par exemple, si on considère  $N = 20$  classifieurs de probabilité d'erreur individuelle  $p = 0.3$ , alors la probabilité qu'au moins 10 classifieurs parmi les 20 fournissent une mauvaise réponse vaut 0,048.

De manière générale, il est possible de déduire la majoration suivante à partir de l'inégalité de Hoeffding (annexe A.1, à faire en exercice!), dans le cas où  $m \geq Np$  :

$$\sum_{k=m}^N \binom{N}{k} p^k (1-p)^{N-k} \leq e^{-2N(m/N-p)^2}$$

Si  $m = \lceil N/2 \rceil$  et  $p < 1/2$ , la condition d'application de l'inégalité est bien vérifiée et on déduit :

$$\sum_{k=\lceil N/2 \rceil}^N \binom{N}{k} p^k (1-p)^{N-k} \leq e^{-2N(p-1/2)^2}$$

Ce majorant décroît effectivement très vite lorsque  $N$  croît dès que  $p < 1/2$ .

On a donc tout intérêt à associer des classifieurs faibles pour en faire un classifieur fort. Attention néanmoins, le classifieur fort construit ici en choisissant la classe majoritaire parmi les classes prédites par les classifieurs faibles n'a l'assurance d'obtenir de bonnes performances que si les classifieurs faibles fournissent des réponses statistiquement indépendantes. Dans le cas contraire, notre calcul n'est plus valable.

Associer les réponses de classifieurs faibles pour construire un classifieur fort est l'objet des « méthodes ensemblistes ». Nous discutons à présent les deux grandes familles de méthodes ensemblistes, les approches de type *bagging* et de type *boosting*.

## 7.2 Techniques de *bagging*

Après avoir expliqué le *bagging* en général, on expliquera l'algorithme inspiré du *bagging* le plus populaire : les forêts aléatoires.

### 7.2.1 Agrégation *bootstrap*

Dans les techniques de *bagging*, on construit des classifieurs faibles en entraînant un classifieur d'une famille donnée sur des sous-ensembles de la base d'apprentissage initiale. Ces sous-ensembles sont appelés échantillons *bootstrap*. Ils sont construits par tirage aléatoire avec remise à partir de la base initiale : ils peuvent donc présenter plusieurs fois la même observation, et les échantillons *bootstrap* peuvent avoir des observations en commun. Le tirage avec remise est supposé assurer que les réponses des classifieurs faibles sont indépendantes, car la construction de chaque échantillon *bootstrap* ne dépend pas des autres échantillons. Chaque échantillon *bootstrap* permet alors de construire un classifieur. Un classifieur fort agrège (combine) ensuite les prédictions de ces différents classifieurs face à une nouvelle observation, par exemple en choisissant la classe majoritaire parmi les classes prédites. Le nom *bagging* vient de de *Bootstrap AGGREGatING*. La figure 7.1 illustre la technique de *bagging*.

Dans la section 2.2 du chapitre 2, nous avons vu le dilemme biais-fluctuation. Lorsqu'on choisit un classifieur dans une famille donnée (une méthode d'apprentissage) par minimisation du risque empirique sur la base d'apprentissage, le risque moyen de prédiction de ce



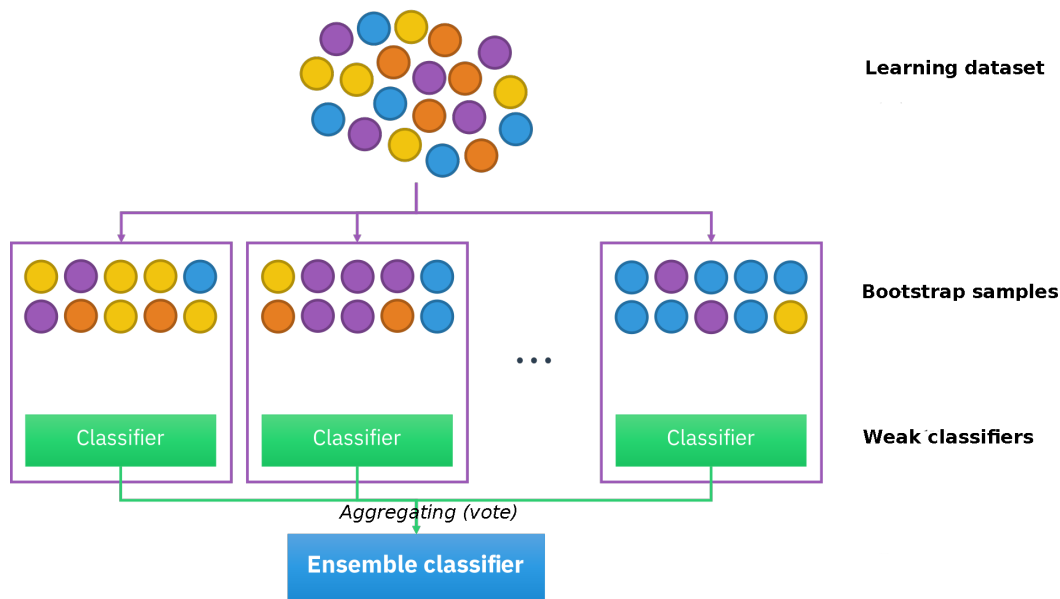


FIGURE 7.1 – Illustration du bagging pour un problème de classification supervisée à quatre classes. Les échantillons bootstrap sont tirés aléatoirement de l'ensemble d'apprentissage (tirage avec remise). Chaque échantillon bootstrap permet d'entraîner un classifieur faible. Le classifieur ensembliste est construit par agrégation des prédictions des classifieurs faibles. Par exemple, face à une nouvelle observation, le classifieur ensembliste pourra prédire la classe majoritaire parmi les classes prédites par les classifieurs faibles. Illustration adaptée de <https://commons.wikimedia.org/w/index.php?curid=85888768> (by Sirakorn - Own work, CC BY-SA 4.0)

classifieur est majoré par la somme du biais (l'erreur de prédiction minimale que peut réaliser tout classifieur de cette famille) et de la fluctuation (une grande fluctuation peut nous mettre en situation de sur-apprentissage).

Le *bagging* est généralement bien adapté aux méthodes d'apprentissage à forte fluctuation. En effet, dans ce cas chaque classifieur faible entraîné sur un échantillon *bootstrap* peut présenter du surapprentissage, donc fortement dépendre de l'échantillon *bootstrap*, mais le vote pour la prédiction majoritaire lissera la dépendance du classifieur fort à la base d'apprentissage originale. Par contre, le *bagging* ne permet pas de réduire le biais, qui est indépendant de la base d'apprentissage.

### 7.2.2 Forêts aléatoires

Les forêts aléatoires (*random forests*) sont une application du *bagging* aux arbres de décision considérés comme classifieurs faibles.

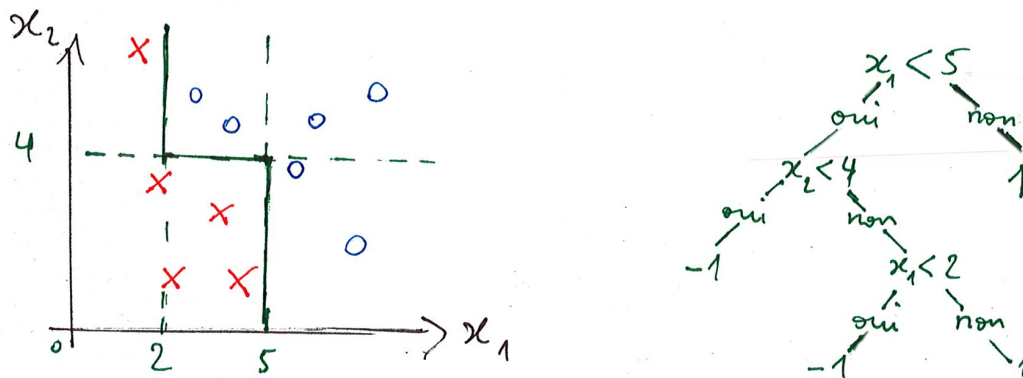


FIGURE 7.2 – Un exemple de classification bi-classe d'observations de  $\mathbb{R}^2$  (caractéristiques  $(x_1, x_2)$ ), et un arbre de décision. À gauche, représentation des observations d'apprentissage : croix rouges (classe -1) et cercles bleus (classe 1). À droite : un arbre binaire de décision de profondeur 3 permettant de classer les observations d'apprentissage. Pour classer une nouvelle observation d'étiquette inconnue, on parcourt l'arbre. La frontière de décision de ce classifieur est représentée en traits pleins verts sur la représentation de gauche. Notons que d'autres arbres de décision pourraient convenir.

### 7.2.2.1 Arbres de décision

Les arbres de décision permettent de faire une prédiction (une classe dans notre cas d'étude) à partir de décisions successives portant sur les caractéristiques d'une observation.

Les arbres de décision sont très populaires car, contrairement à beaucoup des méthodes de l'apprentissage automatique, il est possible d'interpréter les décisions fournies. En effet, ces décisions résultent de la succession de décisions élémentaires sur chaque caractéristique de l'observation dont on cherche à prédire la classe. On parle de « boîte blanche », par opposition à une « boîte noire ». Il s'agit d'une méthode ancienne, développée dans les années 1960. Les médecins (ou garagistes !) utilisent des arbres de décision de manière routinière : la température est-elle supérieure à  $37.5^\circ$  ? si oui, le rythme cardiaque est-il inférieur à 40bpm ? etc., la conclusion de ces tests diagnostiques étant telle ou telle pathologie. La figure 7.2 donne un exemple d'arbre de décision pour un problème de classification d'observations bidimensionnelles.

On dispose d'algorithmes permettant de construire un arbre de décision à partir d'un ensemble d'observations, par exemple l'algorithme CART (*classification and regression trees*) que nous allons décrire. CART construit des arbres binaires de décision : chaque décision est « oui » ou « non », autrement dit avec le vocabulaire de la théorie des graphes, chaque nœud de l'arbre a deux enfants, à l'exception des feuilles. Il s'agit d'un algorithme récursif, qui construit successivement les règles de décision binaires. Chaque règle sera de la forme «  $x_i < T$  » où  $x_i$  est une des caractéristiques de l'observation testée et  $T$  un seuil à déterminer. CART est basé sur un critère de segmentation. Un critère de segmentation est une valeur

calculée sur un sous-ensemble de la base d'apprentissage, qui va permettre de décider sur quelle caractéristique  $x_i$  et quelle valeur  $T$  va se faire la décision en chaque nœud.

On suppose disposer d'un ensemble  $\mathcal{S}$  d'observations. Cet ensemble étant fini, chaque caractéristique  $x_i$  ne prend qu'un nombre fini de valeurs différentes. L'algorithme CART fonctionne de la manière suivante.

1. Pour chaque caractéristique, calculer la valeur du critère de segmentation  $C_{i,j}$  sur l'ensemble des observations vérifiant  $x_i < T_j$ , où  $T_j$  est la  $j$ -ème valeur possible pour  $x_i$ .
2. Sélectionner la caractéristique  $x_{i^*}$  et le seuil  $T_{j^*}$  qui minimise l'ensemble des  $C_{i,j}$ .
3. Séparer l'ensemble d'observations entre le sous-ensemble  $\mathcal{S}_1$  vérifiant  $x_{i^*} < T_{j^*}$ , et le sous-ensemble  $\mathcal{S}_2$  vérifiant  $x_{i^*} \geq T_{j^*}$ .
4. Recommencer à l'étape 1 avec le sous-ensemble  $\mathcal{S}_1$  (respectivement  $\mathcal{S}_2$ ) sauf si  $\mathcal{S}_1$  (respectivement  $\mathcal{S}_2$ ) ne contient que des observations de la même classe.

Le nombre d'observations d'apprentissage étant fini, l'algorithme termine. En algorithmique, on parle d'« algorithme glouton » (*greedy algorithm*).

Notons  $p_k$  la proportion d'observations dans le sous-ensemble courant appartenant à la classe  $k$  ( $k \in \{1, \dots, K\}$  dans le cas de  $K$  classes). Plusieurs critères de segmentation  $C$  sont possibles, les plus utiles sont :

- l'entropie croisée (avec la convention  $0 \log(0) = 0$ ) :

$$C = - \sum_{k=1}^K p_k \log(p_k)$$

- le critère de Gini :

$$C = \sum_{k=1}^K p_k(1 - p_k)$$

Pour ces deux critères,  $C$  est positif, dans le cas biclasse  $C = 0$  si  $p_k = 0$  ou  $p_k = 1$ , et  $C$  est maximal pour  $p_k = 1/2$ . Chaque étape de CART encourage donc la formation de sous-ensembles  $\mathcal{S}_1$  et  $\mathcal{S}_2$  dans lesquels une grande proportion d'observations appartiennent à une seule classe.

La figure 7.3 illustre la formation d'un arbre de décision classifieur. En haut à gauche de la figure, on voit les observations réparties en deux classes (cercles rouges et étoiles bleues) auxquelles on ajoute des observations de test (croix en transparence) pour calculer un score de classification. Toujours en première ligne, on voit aussi les classifications obtenues (classe majoritaire dans l'agrégation des votes) par les arbres de décision obtenus par l'algorithme glouton de construction au fur et à mesure que la profondeur de l'arbre augmente. Ces arbres sont aussi représentés. Pour une profondeur de 3, la région rosée indique qu'une observation  $y$  serait classée comme un cercle rouge, mais la classe des étoiles bleues obtient quelques voix. Pour une profondeur 4, la région blanche indique que les deux classes  $y$  reçoivent le même nombre de votes. Le nombre en bas à droite est la proportion d'observations test bien

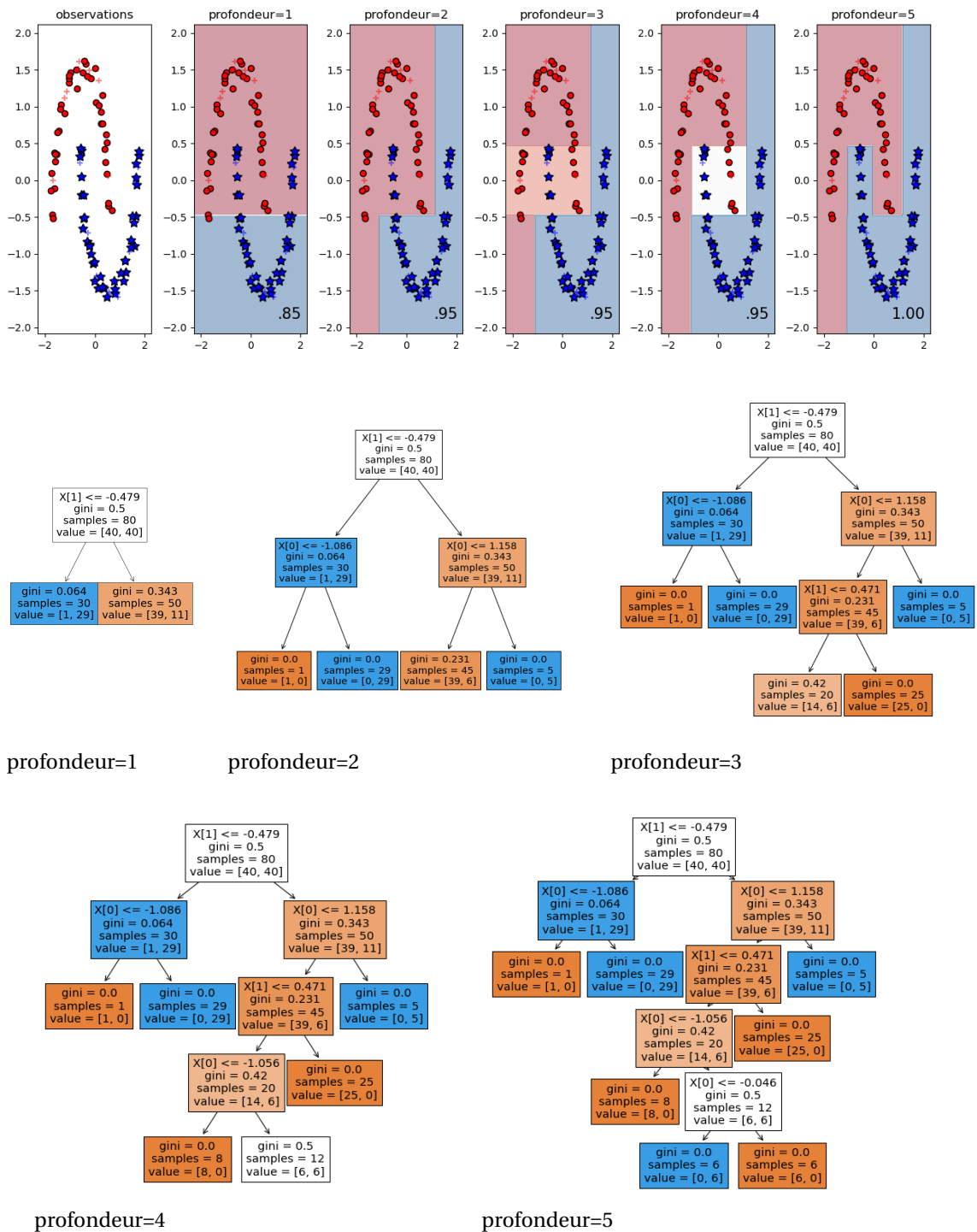


FIGURE 7.3 – Classification de données bidimensionnelles à l'aide de la fonction DecisionTreeClassifier de scikit-learn. Voir le texte.

classées. Les arbres de décision sont également représentés : on indique la règle de décision, la valeur du critère de segmentation (critère de Gini), ainsi que le nombre total d'observations (*samples*) et le nombre d'observation de chaque classe (*value*) dans les sous-arbres du nœud courant. La couleur d'un nœud indique la classe majoritaire dans le sous-arbre. Notons que le critère d'arrêt impose qu'une branche s'arrête de « pousser » dès qu'elle ne contient plus que des observations d'une seule classe.

On a déjà dit que l'interprétabilité des décisions expliquait le succès des arbres de décision. Néanmoins les règles de décision générées dépendent fortement de la base d'observations. En effet, dans CART, les seuils sont choisis parmi les valeurs prises par les caractéristiques. Par ailleurs, les décisions se font uniquement sur les caractéristiques : dans le cas d'observations de  $\mathbb{R}^2$ , la surface de séparation est une union de segments parallèles à l'un ou l'autre des axes. Cette manière de définir l'arbre de décision peut être largement sous-optimale. Pensez par exemple à des observations dans le plan appartenant à deux classes séparées par une droite penchée à 45 degrés : de nombreuses décisions sur les caractéristiques peuvent être nécessaires alors qu'une seule décision combinant linéairement les caractéristiques serait nécessaire.

L'algorithme CART partitionne parfaitement la base d'observation (dans le sens où l'erreur sur la base d'apprentissage est nulle), quitte à ce que l'arbre soit très profond. Pour éviter le sur-apprentissage, on « élague » l'arbre en supprimant les branches concernant trop peu d'observations par exemple. Pour cette raison, les arbres de décision forment généralement des classifieurs faibles.

### 7.2.2.2 *Bagging* et arbres de décision : forêts aléatoires

Les arbres de décision sont susceptibles de fortement dépendre des observations d'apprentissages. Cette famille de classifieurs a donc probablement une forte fluctuation : la technique du *bagging* est par conséquent toute indiquée.

Comme on l'a déjà dit, cette technique est censée bien fonctionner lorsque les classifieurs faibles ont des probabilités d'erreur indépendantes, ce qui motive l'échantillonnage *bootstrap*. Dans le cas des arbres de décision, on introduit un aléa supplémentaire dans l'algorithme de construction expliqué précédemment, en choisissant au hasard  $m$  caractéristiques parmi les  $d$  disponibles pour calculer les valeurs du critère de segmentation, à la place de toutes les caractéristiques disponibles.

L'algorithme de classification de la forêt aléatoire (*random forest*) consiste à construire de tels arbres de décision aléatoires à partir d'échantillons *bootstrap*, puis à sélectionner la classe majoritaire parmi les classes prédites par ces classifieurs faibles. On parle aussi de forêt d'arbres aléatoires. La figure 7.4 montre un exemple de classification par forêt aléatoire.

Remarquons que l'on perd l'interprétabilité des résultats des arbres de décision : les forêts aléatoires sont, elles, des « boîtes noires ».

Les principaux paramètres à régler sont le nombre d'arbres dans la forêt, le nombre de caractéristiques sélectionnées aléatoirement dans la construction des arbres de décision, le critère de segmentation, et la méthode d'élagage des arbres de décision. Pour fixer les idées,

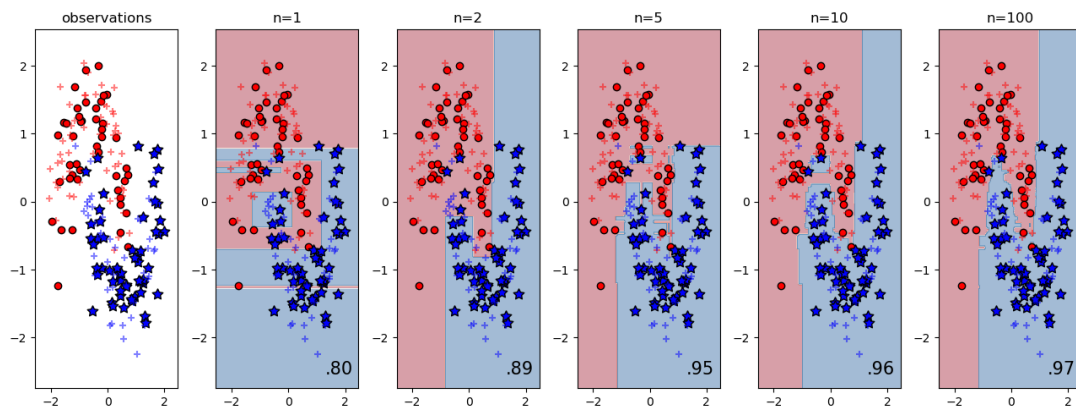


FIGURE 7.4 – Classification de données bidimensionnelles à l'aide de la fonction `RandomForestClassifier` de `scikit-learn`, pour différents nombres  $n$  d'arbres dans la forêt. Les arbres aléatoires sont construits sur une caractéristique ( $m = 1$ ). Les observations sont séparées en deux classes, des observations tests (croix en transparence) permettent de calculer un taux de classifications correctes, indiqué en bas à droite. On voit que plus la forêt contient d'arbres, plus la frontière de classification est complexe.

voici les paramètres par défaut dans `RandomForestClassifier` de `Scikit-learn` : 100 arbres,  $m = \sqrt{d}$ , critère de Gini.

Les forêts aléatoires rencontrent de nombreux succès pratiques. Microsoft l'utilise dans le suivi des mouvements du boîtier Kinect de la console Xbox<sup>1</sup>. Bien entendu, toute une littérature est disponible sur le sujet.

### 7.3 Techniques de *boosting*

Dans les techniques de *boosting*, des classifieurs faibles sont construits séquentiellement sur l'ensemble de la base de données ; à chaque étape un classifieur est construit à partir des classifieurs précédents en pondérant de manière plus importante les observations prédites incorrectement de manière à focaliser les efforts sur ces observations difficiles à classer. La technique du *boosting* est illustrée par la figure 7.5.

#### 7.3.1 Adaboost

L'exemple le plus connu de méthode de type *boosting* est sans doute Adaboost (pour *ADaptive BOOSTing*). Il s'agit d'une technique générale pour construire un classifieur fort à partir de classifieurs faibles. Supposons que l'on dispose d'une famille de classifieurs faible  $\mathcal{H}$ .

1. Voir A. Criminisi, J. Shotton, E. Konukoglu, *Decision forests for classification, regression, density estimation, manifold learning and semi-supervised learning*, Microsoft Research technical report TR-2011-114. [https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/decisionForests\\_MSR\\_TR\\_2011\\_114.pdf](https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/decisionForests_MSR_TR_2011_114.pdf)

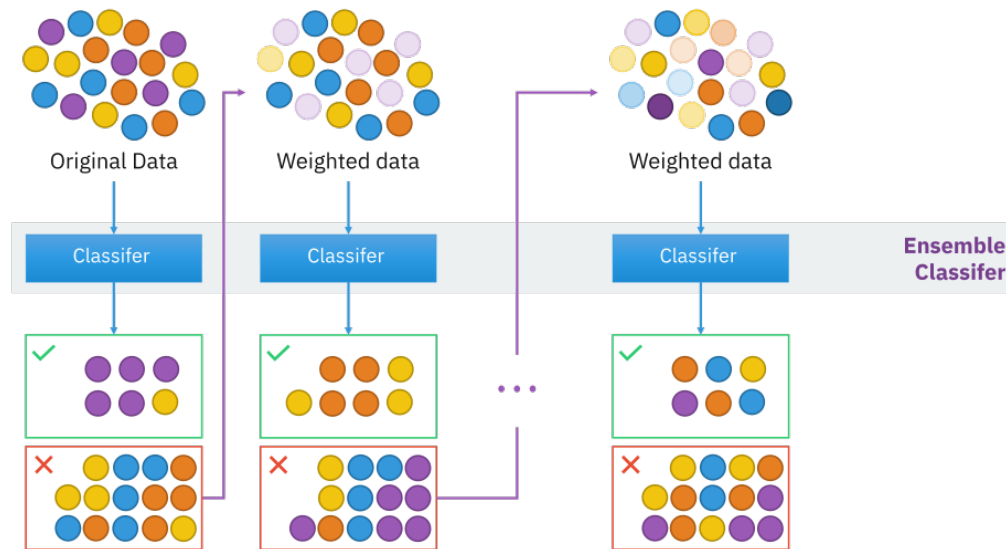


FIGURE 7.5 – Illustration du boosting pour un problème de classification à quatre classes. Un premier classifieur faible est entraîné sur la base d'apprentissage, avec des observations pondérées de manière uniforme. Il sépare cette base entre observations correctement classées et mal classées. Les poids des observations de la base originale sont adaptés de manière à donner un poids plus élevé aux données mal classées. Un second classifieur est entraîné sur ces données pondérées, et ainsi de suite. Ce processus itératif est arrêté après quelques étapes. La réponse du classifieur ensembliste est définie comme une moyenne pondérée des classifieurs faibles intermédiaires. Source de l'illustration : <https://commons.wikimedia.org/w/index.php?curid=85888769> (by Sirakorn - Own work, CC BY-SA 4.0).

On sélectionne dans  $\mathcal{H}$  le classifieur faible minimisant le risque empirique : il commet des erreurs sur la base d'apprentissage. L'idée est de le combiner avec un autre classifieur faible plus performant sur les observations mal classées. Pour se faire, on sélectionne dans  $\mathcal{H}$  le classifieur faible minimisant un risque empirique pondéré, les observations mal classées précédemment ayant un poids plus important. Le nouveau classifieur faible commet toujours des erreurs, et ainsi de suite. La réponse du classifieur ensembliste sera le signe d'une moyenne pondérée des réponses des classifieurs faibles intermédiaires.

Précisons quelques notations avant de détailler Adaboost. On suppose disposer d'une base d'apprentissage de  $N$  observations étiquetées notée  $(\mathbf{x}_i, y_i)_{1 \leq i \leq N}$ , les étiquettes valant -1 ou 1 selon la classe. On note  $h_m$  le classifieur faible sélectionné dans  $\mathcal{H}$  à l'étape  $m$  ( $h_m(\mathbf{x})$  vaut 1 ou -1 selon la classe prédite pour  $\mathbf{x}$ ). La fonction discriminante du classifieur fort construit à la  $m$ -ème étape, combinant les classifieurs faibles précédents, est noté  $K_m$  et vé-

rifie :

$$\begin{cases} K_0 = C_0 \text{ est le classifieur de } \mathcal{H} \text{ minimisant} \\ \text{le nombre d'erreurs pour des observations non pondérées} \\ K_m = K_{m-1} + \alpha_m h_m \end{cases}$$

où  $\alpha_m > 0$ . La règle de décision pour le classifieur ensembliste associé à  $K_m$  est comme souvent d'associer une observation  $x$  à la classe -1 si  $K_m(x) < 0$  et à la classe 1 si  $K_m(x) > 0$ .

Remarquons que le classifieur ensembliste obtenu n'appartient pas *a priori* à la famille de classifieurs faibles  $\mathcal{H}$ . Nous espérons ainsi diminuer le biais d'apprentissage de la famille  $\mathcal{H}$ .

La question est de savoir comment choisir les poids  $\alpha_m$  et les classifieurs  $h_m$  à chaque étape, de manière à obtenir un classifieur fort intéressant après quelques étapes. Nous allons faire les choix de manière à minimiser un risque empirique calculé sur la base d'apprentissage. Dans Adaboost, on introduit le risque empirique du classifieur  $K_m$  associé à une fonction de perte exponentielle (*exponential loss*) :

$$E_m = \sum_{i=1}^N e^{-y_i K_m(x_i)}$$

Il s'agit bien d'une fonction de perte valide :  $e^{-y_i K_m(x_i)}$  est plus grand lorsque  $y_i$  et  $K_m(x_i)$  ont des signes différents (erreur de prédiction de  $K_m$ ) que lorsque  $y_i$  et  $K_m(x_i)$  ont même signe (prédiction correcte). Le choix d'une fonction de perte exponentielle va permettre de mener à bien les calculs suivants. Il est possible d'utiliser une perte logistique (voir section 8.4.1 au chapitre 8), conduisant à l'algorithme LogitBoost.

Par définition de  $K_m$  :

$$E_m = \sum_{i=1}^N e^{-y_i(K_{m-1}(x_i) + \alpha_m h_m(x_i))} = \sum_{i=1}^N w_i^m e^{-y_i \alpha_m h_m(x_i)}$$

où  $w_i^m = e^{-y_i K_{m-1}(x_i)}$ .

Comme  $y_i h_m(x_i) = 1$  si  $h_m(x_i) = y_i$  et  $y_i h_m(x_i) = -1$  si  $h_m(x_i) \neq y_i$ , on peut séparer dans l'expression de  $E_m$  les observations bien et mal classées par le classifieur  $h_m$  et écrire :

$$E_m = \sum_{i \text{ t.q. } h_m(x_i) = y_i} w_i^m e^{-\alpha_m} + \sum_{i \text{ t.q. } h_m(x_i) \neq y_i} w_i^m e^{\alpha_m}$$

soit :

$$E_m = e^{-\alpha_m} \left( \sum_{i=1}^N w_i^m + \sum_{i \text{ t.q. } h_m(x_i) \neq y_i} w_i^m (e^{2\alpha_m} - 1) \right)$$

On voit que seule la seconde somme dépend du choix de  $h_m$ . Minimiser  $E_m$  sur l'ensemble des classifieurs faibles  $\mathcal{H}$  revient donc à minimiser ce terme (car on verra que  $e^{2\alpha_m} - 1 > 0$ ). On sélectionne donc le classifieur faible  $h_m$  qui minimise :  $\sum_{i \text{ t.q. } h_m(x_i) \neq y_i} w_i^m$ , c'est-à-dire la somme des coûts  $w_{i,m}$  sur les observations mal classées.

Remarquons que le choix initial de  $K_0 = C_0$  comme le classifieur de  $\mathcal{H}$  minimisant le nombre d'erreurs revient à dire qu'on a minimisé le risque empirique avec  $w_i^m = 1/N$  pour



tout  $i$  puisque la somme à minimiser est alors proportionnelle au nombre d'observations mal classées.

Il faut également choisir le poids  $\alpha_m$ . On minimise  $E_m$  par rapport à  $\alpha_m$  en annulant la dérivée. Comme :

$$\frac{\partial E_m}{\partial \alpha_m} = -\alpha_m e^{-\alpha_m} \sum_{i \text{ t.q. } h_m(\mathbf{x}_i)=y_i} w_i^m + \alpha_m e^{\alpha_m} \sum_{i \text{ t.q. } h_m(\mathbf{x}_i) \neq y_i} w_i^m$$

cette dérivée s'annule pour :

$$\alpha_m = \frac{1}{2} \log \left( \frac{\sum_{i \text{ t.q. } h_m(\mathbf{x}_i)=y_i} w_i^m}{\sum_{i \text{ t.q. } h_m(\mathbf{x}_i) \neq y_i} w_i^m} \right)$$

Nous avons donc une stratégie pour choisir un classifieur  $h_m$  et mettre à jour son poids  $\alpha_m$ . Concernant la mise à jour des poids des observations  $w_{i,m}$ , on remarque que, par définition de  $K_m$  :

$$w_i^{m+1} = e^{-y_i K_m(\mathbf{x}_i)} = w_i^m e^{-\alpha_m y_i h_m(\mathbf{x}_i)}$$

On déduit la règle de mise à jour :

$$\begin{cases} \text{si } h_m(\mathbf{x}_i) \neq y_i : & w_i^{m+1} = w_i^m e^{\alpha_m} \\ \text{si } h_m(\mathbf{x}_i) = y_i : & w_i^{m+1} = w_i^m e^{-\alpha_m} \end{cases}$$

Dans les implantations logicielles, on normalise les poids à chaque étape de manière à ce que  $\sum_{i=1}^N w_i^m = 1$  pour éviter des problèmes d'instabilité numérique.

En notant  $\varepsilon_m = \sum_{i \text{ t.q. } h_m(\mathbf{x}_i) \neq y_i} w_i^m$  le coût des erreurs, si les poids sont normalisés on a aussi  $1 - \varepsilon_m = \sum_{i \text{ t.q. } h_m(\mathbf{x}_i)=y_i} w_i^m$ , donc :

$$\alpha_m = \frac{1 - \varepsilon_m}{2\varepsilon_m}$$

Au passage,  $e^{2\alpha_m} - 1 = e^{\frac{1-\varepsilon_m}{\varepsilon_m}} - 1 > 0$ . L'algorithme Adaboost s'écrit donc de la manière suivante :

Initialisation :  $\forall 1 \leq i \leq N, w_i^0 = 1/N$ .

Pour  $m = 0$  à  $M$ ,

1. sélectionner le classifieur faible  $h_m$  dans  $\mathcal{H}$  minimisant le coût des erreurs

$$\varepsilon_m = \sum_{i \text{ t.q. } h_m(x_i) \neq y_i} w_i^m$$

2. fixer le poids  $\alpha_m$  de ce classifieur comme :

$$\alpha_m = \frac{1 - \varepsilon_m}{2\varepsilon_m}$$

3. mettre à jour les poids des observations :

$$\forall 1 \leq i \leq N, \begin{cases} \text{si } h_m(x_i) \neq y_i : & w_i^{m+1} = w_i^m e^{\alpha_m} \\ \text{si } h_m(x_i) = y_i : & w_i^{m+1} = w_i^m e^{-\alpha_m} \end{cases}$$

les normaliser tels que  $\sum_{i=1}^N w_i^{m+1} = 1$

4. revenir en 1.

Le nombre d'étapes  $M$  est fixé par l'utilisateur : une trop grande valeur de  $M$  peut conduire à un sur-apprentissage.

Dans Adaboost, les observations sont « stimulées » (*boosted*) par la mise à jour des poids : une observation correctement classée par  $h_m$  à un poids qui diminue à l'étape  $m+1$  (car  $e^{-\alpha_m} < 1$ ) et une observation mal classée à un poids qui augmente (car  $e^{\alpha_m} > 1$ ). Le classifieur faible  $h_{m+1}$  est donc choisi pour se focaliser sur les observations mal classées par les classifieurs faibles précédents.

Le classifieur obtenu par Adaboost combine les classifieurs faibles par un vote pondéré de leurs prédictions. Plus un classifieur faible est performant ( $\varepsilon_m$  faible), plus son poids est élevé ( $\alpha_m$  grand). Par comparaison, le *bagging* combine les réponses par un vote non pondéré.

Nous avons décrit le principe de base d'Adaboost : de nombreuses variantes existent, ainsi qu'une bibliographie foisonnante sur les propriétés théoriques.

Adaboost est souvent utilisé avec des arbres de décisions comme famille  $\mathcal{H}$  de classifieurs faibles, ce qui permet de résoudre l'étape 1 de l'algorithme en un temps raisonnable. Pour ce faire, il suffit de prendre comme critère de segmentation le coût des erreurs. Un choix très répandu est même d'utiliser des « souches de classification » (*decision stumps*) qui sont des arbres de décision de profondeur 1. Ce sont donc des classifieurs qui se contentent de tester si une caractéristique de l'observation est inférieure à un seuil.

Adaboost a rencontré de nombreux succès dans des applications diverses. Il est par exemple utilisé dans l'algorithme de Viola-Jones de détection d'objets (en particulier de visages) dans les images, qui peut fonctionner en temps réel avec de bonnes performances.<sup>2</sup>

La figure 7.6 montre un exemple de classification par Adaboost.

2. Voir les exemples dans l'article original : P. Viola, M. Jones, *Robust real-time object detection*, International Journal of Computer Vision, 57(2), p. 137-154, 2004.

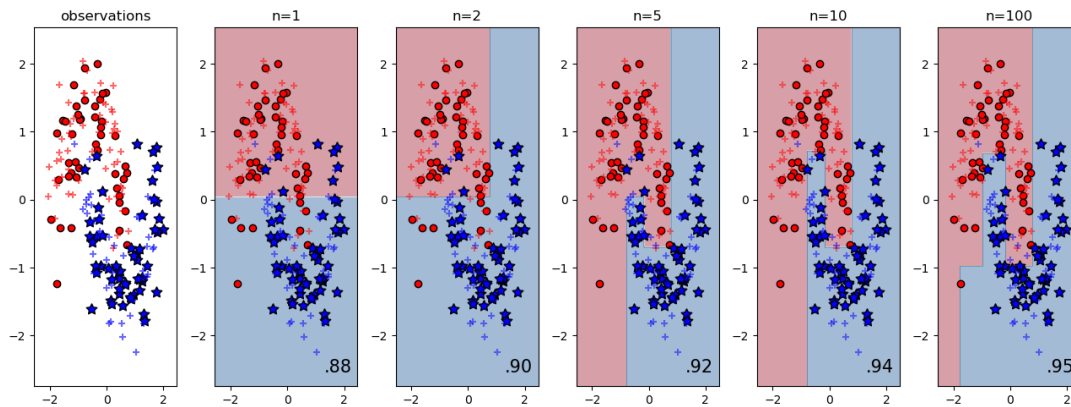


FIGURE 7.6 – Classification de données bidimensionnelles à l'aide de la fonction `AdaBoostClassifier` de `scikit-learn`, pour différents nombres  $n$  de classifieurs faibles. Les classifieurs faibles sont des troncs de classification. Les observations sont séparées en deux classes, des observations tests (croix en transparence) permettent de calculer un taux de classifications correctes, indiqué en bas à droite. Là aussi, plus le nombre de troncs est grands, plus la frontière de classification est complexe.

### 7.3.2 Gradient boosting

Dans notre présentation d'Adaboost, les poids  $\alpha_m$  et classifieurs  $h_m$  sont choisis à chaque étape de manière à minimiser le risque empirique  $E_m$  du classifieur  $K_m$  :

$$E_m = \sum_{i=1}^N \ell(y_i, K_m(\mathbf{x}_i))$$

avec  $\ell$  la fonction de perte exponentielle et où, par définition,  $K_m = K_{m-1} + \alpha_m h_m$ .

Dans les techniques de *gradient boosting*, on généralise cette approche à d'autres fonctions de perte seulement supposées différentiables.

L'idée est de minimiser  $E_m$  en suivant la plus forte pente (*steepest descent*). Un développement limité à l'ordre 1 nous donne :

$$\begin{aligned} E_m &= \sum_{i=1}^N \ell(y_i, K_{m-1}(\mathbf{x}_i) + \alpha_m h_m(\mathbf{x}_i)) \\ &\simeq \sum_{i=1}^N \ell(y_i, K_{m-1}(\mathbf{x}_i)) + \alpha_m \sum_{i=1}^N \frac{\partial \ell}{\partial y}(y_i, K_{m-1}(\mathbf{x}_i)) h_m(\mathbf{x}_i) \\ &= E_{m-1} + \alpha_m \sum_{i=1}^N \frac{\partial \ell}{\partial y}(y_i, K_{m-1}(\mathbf{x}_i)) h_m(\mathbf{x}_i) \end{aligned}$$

où  $\frac{\partial \ell}{\partial y}$  désigne la dérivée partielle de  $\ell$  par rapport à sa deuxième composante.

Pour minimiser  $E_m$  en suivant la plus forte pente, il faut donc :

- choisir  $h_m$  dans la classe  $\mathcal{H}$  de manière à minimiser  $-\sum_{i=1}^N \frac{\partial \ell}{\partial y}(y_i, K_{m-1}(\mathbf{x}_i)) h_m(\mathbf{x}_i)$  qui est un produit scalaire entre vecteurs de dimension  $N$ ,
- choisir  $\alpha_m > 0$  minimisant  $E_m$  (problème d'optimisation 1D, cf *line search* dans les algorithmes de descente).

Selon le choix de la fonction de perte  $\ell$ , le *gradient boosting* s'adapte à des problèmes de classification ou de régression.

XGBoost (*eXtreme Gradient Boosting*), « extrêmement » populaire parmi les participants aux défis Kaggle pour ses très bonnes performances pratiques, est un exemple de *gradient boosting* où les classifieurs faibles sont des arbres de décision ou de régression, qui consiste à utiliser une approximation d'ordre 2 plutôt que d'ordre 1 dans la minimisation, ainsi qu'un terme de régularisation afin d'éviter le surapprentissage. Plus de détails ici :

T. Chen, C. Guestrin. *XGBoost : A scalable tree boosting system*. Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), 2016.

L'implémentation de référence est là :

<https://github.com/dmlc/xgboost>

## 7.4 Pour approfondir...

**Régression** *Bagging* et *boosting* s'appliquent également sans difficulté à des problèmes de régression. Les arbres de décision deviennent des arbres de régression qui permettent de construire des fonctions de régression constantes par morceaux.

**Un peu d'étymologie** Le *bootstrapping* est une technique d'utilisation générale en statistique. Selon Bradley Efron, qui a introduit cette technique dans un article de 1979 :

*"Its name celebrates Baron Munchausen's success in pulling himself up by his own bootstraps from the bottom of a lake."*

(les *bootstraps* sont les anneaux de cuir ou de tissu qui aident à enfiler les bottes ou certaines chaussures).

Voir la page 177 de :

B. Efron and T. Hastie, *Computer age statistical inference*, Cambridge University Press, 2016.

## Machines à vecteurs supports

Dans notre cours, nous discutons des classifieurs linéaires comme le classifieur de la régression logistique au chapitre 6, ou le perceptron de Rosenblatt dans le prochain chapitre. Dans le cas biclasse, la surface de séparation est un hyperplan de l'espace des observations. Lorsque les deux classes sont bien séparées (sans erreur) par un tel hyperplan, on dit qu'elles sont linéairement séparables. Ce chapitre traite des machines à vecteurs supports (*Support Vector Machines*, SVM), dont la version la plus simple fait également partie de la famille des classifieurs linéaires. Elles se prêtent bien à l'« astuce du noyau » qui leur permettent de s'adapter à des données non linéairement séparables. Les SVM à noyau sont les classifieurs phares de la fin des années 1990 et début des années 2000 et ont rencontré de nombreux succès. Elles sont toujours utilisées, en particulier lorsqu'on dispose de données en quantité limitée.

### 8.1 Notion de marge maximale

On suppose disposer d'un jeu d'observations réparties en deux classes linéairement séparables. Dans ce chapitre, les  $N$  observations étiquetées de la base d'apprentissage seront notées  $(\mathbf{x}_n, y_n)$ , avec  $y_n \in \{-1, 1\}$ . Les hyperplans séparateurs de paramètres  $(\mathbf{w}, b)$  vérifient, pour tout  $n$  entre 1 et  $N$  :

$$\begin{cases} \mathbf{w} \cdot \mathbf{x}_n + b > 0 & \text{si } y_n = 1 \\ \mathbf{w} \cdot \mathbf{x}_n + b < 0 & \text{si } y_n = -1 \end{cases}$$

Ainsi, pour tout  $n$ ,  $y_n(\mathbf{w} \cdot \mathbf{x}_n + b) = |\mathbf{w} \cdot \mathbf{x}_n + b|$ . Parmi ces hyperplans séparateurs, on s'intéresse à celui qui est le plus éloigné des points des deux classes.

Rappelons l'expression de la distance d'un point à un plan.

**Lemme 2** Si un hyperplan a pour équation  $\mathbf{w} \cdot \mathbf{x} + b = 0$ , la distance euclidienne d'un point  $\mathbf{x}^*$  à ce plan est :  $|\mathbf{w} \cdot \mathbf{x}^* + b| / \|\mathbf{w}\|_2$ .

*Démonstration*

En effet, le point de l'hyperplan le plus proche de  $\mathbf{x}^*$  est sa projection orthogonale, qui

s'écrit sous la forme  $\mathbf{x}^* + \lambda \mathbf{w}$  où  $\lambda \in \mathbb{R}$  car  $\mathbf{w}$  est orthogonal à l'hyperplan. Comme cette projection appartient à l'hyperplan,  $\lambda$  vérifie  $\mathbf{w} \cdot (\mathbf{x}^* + \lambda \mathbf{w}) + b = 0$ , soit  $\lambda = -(b + \mathbf{w} \cdot \mathbf{x}^*) / \|\mathbf{w}\|_2^2$ . Ainsi, la distance d'un point  $\mathbf{x}^*$  à sa projection orthogonale est  $\|\mathbf{x}^* + \lambda \mathbf{w} - \mathbf{x}^*\|_2 = |\lambda| \|\mathbf{w}\|_2 = |\mathbf{w} \cdot \mathbf{x}^* + b| / \|\mathbf{w}\|_2$ .  $\square$

La plus petite distance entre les observations et un hyperplan séparateur de paramètres  $(\mathbf{w}, b)$  est donc :

$$\frac{1}{\|\mathbf{w}\|_2} \min_{1 \leq n \leq N} y_n(\mathbf{w} \cdot \mathbf{x}_n + b)$$

On appelle « marge » de l'hyperplan séparateur le double de cette quantité.

Par conséquent, les paramètres du séparateur de plus grande marge sont définis par :

$$\arg \max_{\mathbf{w}, b} \left\{ \frac{1}{\|\mathbf{w}\|_2} \min_{1 \leq n \leq N} y_n(\mathbf{w} \cdot \mathbf{x}_n + b) \right\}$$

Notons que si les deux classes sont linéairement séparables, la solution de ce problème de maximisation est bien un hyperplan séparateur car les hyperplans qui ne sont pas séparateurs sont tels que  $\min_n y_n(\mathbf{w} \cdot \mathbf{x}_n + b) < 0$  (car au moins un point n'est pas correctement classé), au contraire des hyperplans séparateurs pour lesquels  $\min_n y_n(\mathbf{w} \cdot \mathbf{x}_n + b) \geq 0$  pour tout  $n$ .

Le séparateur de plus grande marge définit un classifieur : si  $\mathbf{w} \cdot \mathbf{x} + b > 0$  alors l'observation  $\mathbf{x}$  est classée dans 1, et dans  $-1$  sinon.

Les observations  $\mathbf{x}_n$  qui réalisent le minimum dans  $\min_n y_n(\mathbf{w} \cdot \mathbf{x}_n + b)$  sont appelés « vecteurs de support » ou « vecteurs supports ». On reviendra sur cette notion plus tard.

Les séparateurs à vaste marge, dont SVM est l'acronyme en français, ainsi définis sont appelés machines à vecteurs supports (*support vector machines*, SVM).

Marge, hyperplan séparateur et vecteurs supports sont illustrés par la figure 8.1.

### 8.1.1 Problème d'optimisation sous contraintes associé

La distance d'un point à un hyperplan est invariante par multiplication de  $\mathbf{w}$  et  $b$  par un scalaire non nul, comme on le voit sur la formule rappelée en introduction. On suppose donc sans perte de généralité que les observations  $\mathbf{x}_n$  les plus proches du plan (les vecteurs de support) vérifient :

$$y_n(\mathbf{w} \cdot \mathbf{x}_n + b) = 1$$

Le problème de recherche des paramètres  $(\mathbf{w}, b)$  du séparateur à plus grande marge revient alors à celui de maximiser  $1/\|\mathbf{w}\|_2$ , ou de manière équivalente, à résoudre le problème de minimisation suivant :

$$(primal\ 1) \quad \begin{cases} \arg \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|_2^2 \\ \text{sous contraintes : } \forall 1 \leq n \leq N, y_n(\mathbf{w} \cdot \mathbf{x}_n + b) \geq 1 \end{cases}$$

Ce problème est qualifié de primal. Notons que chaque observation étiquetée  $(\mathbf{x}_n, y_n)$  induit une contrainte.

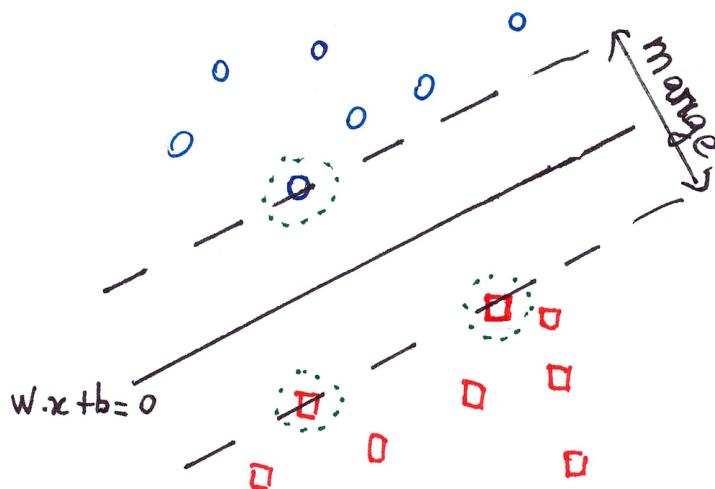


FIGURE 8.1 – Discrimination entre deux classes : marge d'une droite séparatrice d'équation  $\mathbf{w} \cdot \mathbf{x} + b = 0$  et vecteurs supports entourés.

Il s'agit d'un problème convexe (voir annexe B.1), appartenant à la famille des problèmes de programmation quadratique car la fonction objectif est quadratique et les contraintes sont linéaires. Le problème admet donc une unique solution.

Nous allons exprimer le dual de ce problème d'optimisation sous contraintes.

## 8.1.2 Dualité pour le problème des SVM

### 8.1.2.1 Dual de Wolfe

Dans le problème (*primal 1*), la fonction objectif et les contraintes sont convexes et différentiables. On peut donc appliquer les résultats de l'annexe B.2 sur la dualité de Wolfe.

Dans le problème des SVM,  $(\mathbf{w}, b)$  joue le rôle de la variable  $\mathbf{x}$  de l'annexe B.2, les contraintes sont  $g_n(\mathbf{x}) = -(y_n(\mathbf{w} \cdot \mathbf{x}_n + b) - 1)$ , et le Lagrangien est :

$$L(\mathbf{w}, b, \boldsymbol{\lambda}) = \frac{1}{2} \|\mathbf{w}\|_2^2 - \sum_{n=1}^N \lambda_n (y_n(\mathbf{w} \cdot \mathbf{x}_n + b) - 1)$$

D'après les conditions Karush, Kuhn et Tucker (KKT), nécessaires et suffisantes ici,  $(\mathbf{w}^*, b^*)$  est solution optimale du primal si et seulement s'il existe  $\boldsymbol{\lambda}^* = (\lambda_1, \lambda_2, \dots, \lambda_N)$  où  $\forall 1 \leq i \leq N$ ,  $\lambda_i \geq 0$ , tel que  $(\mathbf{w}^*, b^*)$  vérifie :

$$\forall 1 \leq n \leq N, \lambda_n^* (y_n(\mathbf{w}^* \cdot \mathbf{x}_n + b^*) - 1) = 0$$

et  $\boldsymbol{\lambda}^*$  maximise  $L(\mathbf{w}^*, b^*, \boldsymbol{\lambda})$  sous les contraintes  $\frac{\partial L}{\partial \mathbf{w}} = 0$  et  $\frac{\partial L}{\partial b} = 0$ , ce qui est respectivement équivalent à

$$\mathbf{w}^* = \sum_{n=1}^N \lambda_n y_n \mathbf{x}_n$$

et

$$\sum_{n=1}^N \lambda_n y_n = 0$$

On obtient :

$$L(\mathbf{w}^*, b^*, \boldsymbol{\lambda}) = \frac{1}{2} \left\| \sum_{n=1}^N \lambda_n y_n \mathbf{x}_n \right\|_2^2 - \sum_{n=1}^N \lambda_n \left( y_n \left( \left( \sum_{m=1}^N \lambda_m y_m \mathbf{x}_m \right) \cdot \mathbf{x}_n + b^* \right) - 1 \right)$$

Or :

$$\left\| \sum_{n=1}^N \lambda_n y_n \mathbf{x}_n \right\|_2^2 = \sum_{n=1}^N \sum_{m=1}^N \lambda_n \lambda_m y_n y_m \mathbf{x}_n \cdot \mathbf{x}_m$$

et :

$$\sum_{n=1}^N \lambda_n \left( y_n \left( \left( \sum_{m=1}^N \lambda_m y_m \mathbf{x}_m \right) \cdot \mathbf{x}_n + b^* \right) - 1 \right) = \sum_{n=1}^N \sum_{m=1}^N \lambda_n \lambda_m y_n y_m \mathbf{x}_n \cdot \mathbf{x}_m + b^* \sum_{n=1}^N \lambda_n y_n - \sum_{n=1}^N \lambda_n$$

Comme  $\sum_{n=1}^N \lambda_n y_n = 0$ , on simplifie l'expression du Lagrangien en :

$$L(\mathbf{w}^*, b^*, \boldsymbol{\lambda}) = \sum_{n=1}^N \lambda_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \lambda_n \lambda_m y_n y_m \mathbf{x}_n \cdot \mathbf{x}_m$$

Le problème dual de Wolfe consiste donc à résoudre :

$$(dual\ 1) \quad \begin{cases} \arg \max_{\boldsymbol{\lambda}} \sum_{n=1}^N \lambda_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \lambda_n \lambda_m y_n y_m \mathbf{x}_n \cdot \mathbf{x}_m \\ \text{sous contraintes : } \forall 1 \leq n \leq N, \lambda_n \geq 0 \text{ et } \sum_{n=1}^N \lambda_n y_n = 0 \end{cases}$$

À l'optimum, les conditions KKT nous fournissent  $\forall n, \lambda_n^* (y_n (\mathbf{w}^* \cdot \mathbf{x}_n + b^*) - 1) = 0$ .

Cela signifie, pour tout  $n \in \{1, \dots, N\}$  :

- soit  $\mathbf{w}^* \cdot \mathbf{x}_n + b^* = y_n$  (et  $\lambda_n^* \geq 0$ ) : l'observation  $\mathbf{x}_n$  est un vecteur support
- soit  $\lambda_n^* = 0$

Or  $\mathbf{w}^* = \sum_{n=1}^N \lambda_n^* y_n \mathbf{x}_n$  : l'hyperplan séparateur optimal ne dépend donc que des vecteurs supports. Les autres observations ont une contribution nulle car pour les « vecteurs non-supports »,  $\lambda_n^* = 0$ .

### 8.1.3 Résumé

Le problème primal des SVM s'écrit :

$$(primal\ 1) \quad \begin{cases} \arg \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|_2^2 \\ \text{sous contraintes : } \forall 1 \leq n \leq N, y_n (\mathbf{w} \cdot \mathbf{x}_n + b) \geq 1 \end{cases}$$



Le problème dual des SVM s'écrit :

$$(dual\ 1) \quad \left\{ \begin{array}{l} \operatorname{argmax}_{\lambda} \sum_{n=1}^N \lambda_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \lambda_n \lambda_m y_n y_m \mathbf{x}_n \cdot \mathbf{x}_m \\ \text{sous contraintes : } \forall 1 \leq n \leq N, \lambda_n \geq 0 \text{ et } \sum_{n=1}^N \lambda_n y_n = 0 \end{array} \right.$$

Dans les deux cas, il s'agit de programmes quadratiques. Le primal admet  $d+1$  inconnues et  $N$  contraintes, le dual  $N$  inconnues et une contrainte en plus des contraintes de signes sur les  $\lambda_n$ .

Résoudre le problème dual nous fournit  $\lambda^*$ , ensuite on déduit  $\mathbf{w}^*$  par l'expression :

$$\mathbf{w}^* = \sum_{n=1}^N \lambda_n^* y_n \mathbf{x}_n$$

où les termes non nuls dans cette somme sont ceux relatifs aux vecteurs supports. Les observations qui ne sont pas vecteurs supports ne participent pas à la définition de l'hyperplan optimal. Ce résultat est cohérent avec l'interprétation géométrique du problème : sur la figure 8.1, la position des observations qui ne sont pas des vecteurs supports n'influence pas la position du séparateur de plus grande marge.

Enfin, on calcule  $b^*$  à partir des vecteurs supports sachant que pour tout vecteur support,  $\mathbf{w}^* \cdot \mathbf{x}_n + b^* = y_n$ . On peut bien estimer numériquement  $b^*$ , par exemple en calculant la moyenne sur les vecteurs supports  $\mathbf{x}_n$  des valeurs de  $y_n - \mathbf{w}^* \cdot \mathbf{x}_n$ .

## 8.2 Marge souple et variables d'écart

Dans le modèle précédent (dit à marge dure, *hard margin*), les deux classes sont supposées linéairement séparables. Si ce n'est pas le cas, le problème d'optimisation primal n'a pas de solution car l'ensemble des solutions réalisables est vide. Pour tenir compte de perturbations sur les observations qui pourraient rendre les classes non-linéairement séparables, ou permettre des erreurs de classification sur la base d'apprentissage afin d'éviter le surapprentissage, on introduit un modèle de SVM à marge souple (*soft margin*).

Une variable d'écart à la marge (*slack variable*) pour l'observation  $\mathbf{x}_n$  est un réel  $\xi_n > 0$  tel que :

$$\xi_n = \begin{cases} 0 & \text{si l'observation } \mathbf{x}_n \text{ satisfait la contrainte sur la marge, i.e. } y_n(\mathbf{w} \cdot \mathbf{x}_n + b) \geq 1 \\ |y_n - (\mathbf{w} \cdot \mathbf{x}_n + b)| & \text{sinon} \end{cases}$$

Dans tous les cas, comme  $y_n \in \{-1, 1\}$ , la variable d'écart de la  $n$ -ème observation s'exprime ainsi :

$$\xi_n = \max\{0, 1 - y_n(\mathbf{w} \cdot \mathbf{x}_n + b)\}$$

La notion de variable d'écart est illustrée par la figure 8.2.

On va estimer les paramètres d'une SVM à marge souple par optimisation, en réalisant un compromis entre, d'une part, la marge  $1/\|\mathbf{w}\|$  que l'on veut la plus large possible, et, d'autre part, les écarts qui prennent des valeurs strictement positives.

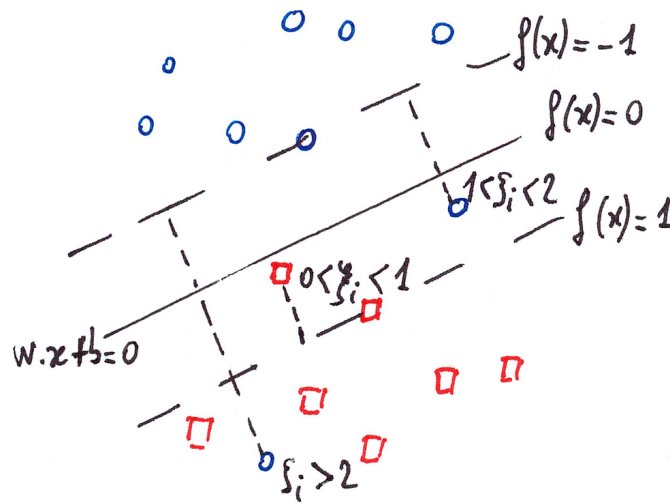


FIGURE 8.2 – SVM : marge souple et variables d'écart. Notons les valeurs prises par  $\xi_i$  selon la position de l'observation  $x_i$  par rapport à la droite de séparation. Ici,  $f(x) = \mathbf{w} \cdot \mathbf{x} + b$ .

### 8.2.1 Problème primal

Le problème primal de la SVM à marge souple s'écrit :

$$(primal\ 2) \begin{cases} \operatorname{argmin}_{\mathbf{w}, b, \xi} \frac{1}{2} \|\mathbf{w}\|_2^2 + C \sum_{n=1}^N \xi_n \\ \text{sous contraintes : } \forall 1 \leq n \leq N, \xi_n \geq 1 - y_n(\mathbf{w} \cdot \mathbf{x}_n + b) \text{ et } \xi_n \geq 0 \end{cases}$$

Il s'agit toujours d'un problème d'optimisation quadratique.

Le rôle de l'hyperparamètre  $C > 0$  est le suivant :

- $C$  grand : on pénalise fortement les variables d'écart. À l'optimum, les  $\xi_n$  vont donc être petits, si possible nuls : peu d'observations seront mal classées. On risque le sur-apprentissage.
- $C$  proche de 0 : à l'optimum les écarts  $\xi_n$  peuvent être assez grands, de manière à maximiser la marge  $1/\|\mathbf{w}\|_2$  qui a un poids relatif plus important que les écarts dans la fonction objectif. On risque le sous-apprentissage.

### 8.2.2 Problème dual

Le Lagrangien de ce problème s'écrit

$$L(\mathbf{w}, b, \xi, \lambda, \mu) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{n=1}^N \xi_n - \sum_{n=1}^N \lambda_n (y_n(\mathbf{w} \cdot \mathbf{x}_n + b) - 1 + \xi_n) - \sum_{n=1}^N \mu_n \xi_n$$

où pour tout  $n$ ,  $\lambda_n \geq 0$  et  $\mu_n \geq 0$ .

La dérivée partielle du Lagrangien par rapport à  $\xi_n$  fournit  $C - \lambda_n - \mu_n = 0$ , donc le Lagrangien se simplifie en

$$L(\mathbf{w}, b, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{n=1}^N \lambda_n (y_n (\mathbf{w} \cdot \mathbf{x}_n + b) - 1)$$

(les  $\xi_n$  n'apparaissent plus), et  $\lambda_n = C - \mu_n$  avec  $\mu_n \geq 0$  implique  $0 \leq \lambda_n \leq C$ .

Le problème dual de Wolfe est donc :

$$(dual\ 2) \quad \left\{ \begin{array}{l} \arg \max_{\boldsymbol{\lambda}} \sum_{n=1}^N \lambda_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \lambda_n \lambda_m y_n y_m \mathbf{x}_n \cdot \mathbf{x}_m \\ \text{sous contraintes : } \forall 1 \leq n \leq N, 0 \leq \lambda_n \leq C \text{ et } \sum_{n=1}^N \lambda_n y_n = 0 \end{array} \right.$$

On remarque que la seule différence avec le problème *(dual 1)* est la borne supérieure des  $\lambda_n$ , égale à  $C$ . L'introduction des variables d'écart agit comme une régularisation : on contraint l'espace des solutions possibles du dual.

Par ailleurs, on a toujours la relation  $\mathbf{w} = \sum_n \lambda_n y_n \mathbf{x}_n$  :  $\mathbf{w}$  ne dépend que des vecteurs supports (qui sont tels que  $\lambda_n \neq 0$ ), pour lesquels la contrainte correspondante est saturée, ce qui s'écrit :  $y_n (\mathbf{w} \cdot \mathbf{x}_n + b) = 1 - \xi_n$ . Ainsi, les vecteurs supports sont les observations  $\mathbf{x}_n$  vérifiant :

- soit  $\xi_n = 0$  et alors  $y_n (\mathbf{w} \cdot \mathbf{x}_n + b) = 1$  :  $\mathbf{x}_n$  est sur la frontière de la marge,
- soit  $\xi_n > 0$  et  $\mathbf{x}_n$  est à l'« intérieur » de la marge, avec deux possibilités illustrées par la figure 8.2 :
  - si  $0 < \xi_n < 1$ , alors  $y_n (\mathbf{w} \cdot \mathbf{x}_n + b) > 0$  et  $\mathbf{x}_n$  est bien classée
  - si  $\xi_n > 1$ , alors  $y_n (\mathbf{w} \cdot \mathbf{x}_n + b) < 0$  et  $\mathbf{x}_n$  est mal classée

Une remarque très importante à ce stade est que le problème dual ne fait apparaître les observations que par l'intermédiaire de produits scalaires  $\mathbf{x}_n \cdot \mathbf{x}_m$ . Ceci justifie l'introduction de « l'astuce du noyau ».

### 8.3 Astuce du noyau

D'une part, le problème dual des SVM (à marge dure ou souple) ne fait intervenir les observations que par l'intermédiaire de leur produit scalaire.

D'autre part, nous allons voir que des problèmes de classification non linéairement séparables peuvent le devenir lorsqu'on plonge les observations dans un espace vectoriel de dimension plus grande que l'espace initial.

Ces deux remarques justifient la mise en œuvre de « l'astuce du noyau » (*kernel trick*), qui permet de transformer les machines à vecteurs supports en classificateurs non-linéaires.



FIGURE 8.3 – Séparation de points du plan par une droite. Trois points sont toujours séparables quelles que soient les étiquettes. Par contre, certains quadruplets ne sont pas séparables, comme montré à droite.

### 8.3.1 Séparation linéaire et dimension

La motivation de l'utilisation des noyaux est basée sur le constat suivant : il est en un certain sens plus facile de séparer des observations vectorielles par des hyperplans lorsque la dimension de l'espace est grande.

Par exemple, si on considère trois points du plan non alignés, on pourra toujours trouver une droite les séparant, quelles que soient leurs étiquettes. Par contre quatre points du plan ne sont pas toujours séparables selon une droite. Voir la figure 8.3.

La proposition suivante montre que la propriété est générale.

**Proposition 8.1** Dans  $\mathbb{R}^d$ , tout ensemble de  $d + 1$  observations étiquetées  $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_{d+1}, y_{d+1})$  telles que  $(\mathbf{x}_n - \mathbf{x}_{d+1})_{1 \leq n \leq d}$  forme une famille libre est linéairement séparable.

Le fait que les  $d$  vecteurs  $(\mathbf{x}_n - \mathbf{x}_{d+1})$  forment une famille libre est le pendant de la condition « les points ne sont pas alignés » dans l'exemple de la figure 8.3 en dimension 2.

#### Démonstration

Notons  $A$  la matrice dont les colonnes sont formées des vecteurs  $\mathbf{x}_n - \mathbf{x}_{d+1}$ . La matrice  $A$  admet une décomposition  $A = QR$ , où  $Q$  est une matrice orthogonale et  $R$  une matrice triangulaire supérieure, inversible comme  $A$ .

Considérons alors la fonction  $f$  affine en  $\mathbf{x}$  d'équation :

$$f(\mathbf{x}) = \sum_{n=1}^d (y_n - y_{d+1}) (R^{-1}(\mathbf{x}_n - \mathbf{x}_{d+1}))^T R^{-1}(\mathbf{x} - \mathbf{x}_{d+1}) + y_{d+1}$$

Comme  $R^{-1}(\mathbf{x}_n - \mathbf{x}_{d+1})$  est la  $n$ -ème colonne de la matrice  $Q$  qui est orthogonale,

$$(R^{-1}(\mathbf{x}_n - \mathbf{x}_{d+1}))^T R^{-1}(\mathbf{x}_m - \mathbf{x}_{d+1}) = \begin{cases} 0 & \text{si } n \neq m \\ 1 & \text{si } n = m \end{cases}$$

Donc pour tout  $n$  entre 1 et  $d + 1$ ,  $f(\mathbf{x}_n) = y_n$ . L'hyperplan d'équation  $f(\mathbf{x}) = 0$  sépare bien les points, quelles que soient les valeurs des étiquettes.  $\square$

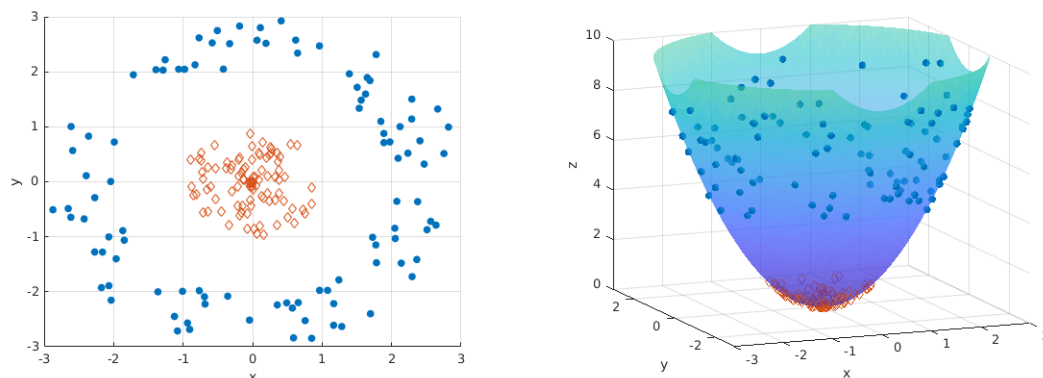


FIGURE 8.4 – Exemple de plongement de points de  $\mathbb{R}^2$  dans  $\mathbb{R}^3$ . Le plongement consiste à projeter les points du plan sur un parabolöide de révolution de  $\mathbb{R}^3$ . Sur le parabolöide dans  $\mathbb{R}^3$  les deux classes sont linéairement séparables, ce qui n'était pas le cas dans  $\mathbb{R}^2$ .

Il est possible de démontrer que pour tout ensemble de  $d+2$  points de  $\mathbb{R}^d$ , certaines affectations d'étiquettes rendent impossible la séparation, comme dans l'exemple de la figure 8.3 pour  $d = 2$ . Au passage, et sans entrer dans les détails, ces deux propriétés permettent d'affirmer que la dimension de Vapnik-Chervonenkis (évoquée au chapitre 2, section 2.3) des hyperplans de  $\mathbb{R}^d$  est  $d + 1$ .

Il est donc plus « facile » de séparer linéairement deux classes lorsque la dimension est grande, en particulier lorsque la dimension est supérieure au nombre d'observations. On peut également interpréter la proposition 8.1 comme une justification de l'utilisation des classifieurs linéaires (régression logistique, perceptron de Rosenblatt, SVM linéaire) lorsque les observations appartiennent à un espace de grande dimension, comme dans le cas de la classification de textes ou d'images.

La figure 8.4 illustre d'une autre manière cette propriété : des observations non linéairement séparables le deviennent lorsqu'on les plonge dans un espace vectoriel de dimension supérieure.

L'« astuce du noyau » (*kernel trick*) utilisée dans les SVM fait qu'on n'aura pas besoin de déterminer explicitement le plongement  $\phi$ , on aura seulement besoin des valeurs des produits scalaires  $\phi(\mathbf{x}) \cdot \phi(\mathbf{y})$ .

### 8.3.2 L'astuce du noyau dans les SVM

On plonge les observations  $\mathbf{x}_n \in \mathbb{R}^d$  dans un espace vectoriel  $\mathcal{H}$  de dimension  $d'$  par  $\phi$ . Cet espace  $\mathcal{H}$  est appelé « espace de redescription » ou *feature space*. D'après la discussion précédente, on espère qu'il sera plus facile de séparer linéairement dans  $\mathcal{H}$ . Il y a donc un risque fort de sur-apprentissage : il est important de considérer le modèle de SVM à marge

douce qui introduit les écarts à la marge (et permet des erreurs de classifications) afin d'éviter le sur-apprentissage.

Nous avons établi plus haut la formulation duale de la SVM séparant les observations une fois celles-ci plongées dans  $\mathcal{H}$  par  $\phi$  :

$$(dual\ 2') \quad \left\{ \begin{array}{l} \arg \max_{\lambda} \sum_{n=1}^N \lambda_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \lambda_n \lambda_m y_n y_m \phi(\mathbf{x}_n) \cdot \phi(\mathbf{x}_m) \\ \text{sous contraintes : } \forall 1 \leq n \leq N, 0 \leq \lambda_n \leq C \text{ et } \sum_{n=1}^N \lambda_n y_n = 0 \end{array} \right.$$

Remarquons que seuls des produits scalaires  $\phi(\mathbf{x}_n) \cdot \phi(\mathbf{x}_m)$  apparaissent. Si on définit le noyau  $k$  tel que  $k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{y})$ , le problème devient :

$$(dual\ 3) \quad \left\{ \begin{array}{l} \arg \max_{\lambda} \sum_{n=1}^N \lambda_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \lambda_n \lambda_m y_n y_m k(\mathbf{x}_n, \mathbf{x}_m) \\ \text{sous contraintes : } \forall 1 \leq n \leq N, 0 \leq \lambda_n \leq C \text{ et } \sum_{n=1}^N \lambda_n y_n = 0 \end{array} \right.$$

Le classifieur SVM s'écrit  $f(\mathbf{x}) = \mathbf{w} \cdot \phi(\mathbf{x}) + b$  avec  $\mathbf{w} = \sum_n \lambda_n y_n \phi(\mathbf{x}_n)$ , où seuls les  $\lambda_n$  relatifs aux vecteurs supports sont non-nuls. Ainsi,  $f(\mathbf{x}) = \sum_{n=1}^N \lambda_n y_n \phi(\mathbf{x}_n) \cdot \phi(\mathbf{x}) + b$ . Le classifieur s'écrit donc uniquement en fonction du noyau  $k$  sous la forme suivante :

$$f(\mathbf{x}) = \sum_{n=1}^N \lambda_n y_n k(\mathbf{x}_n, \mathbf{x}) + b$$

Notons qu'on n'a pas besoin de l'expression de  $\mathbf{w}$ , vecteur normal à l'hyperplan séparateur dans l'espace de redescription, mais seulement de calculer les  $k(\mathbf{x}_n, \mathbf{x})$  pour les vecteurs supports  $\mathbf{x}_n$ . En dimension  $d'$  grande, il est bien plus rapide de calculer  $f(\mathbf{x})$  grâce au noyau et aux vecteurs supports (cela nécessite une somme avec autant de termes que de vecteurs supports, qui sont généralement en nombre limité) qu'en utilisant le calcul explicite du produit scalaire avec  $\mathbf{w}$  (qui nécessite au moins  $d'$  produits).

Que ce soit pour trouver les  $\lambda_n$  ou ensuite pour classifier de nouvelles observations, nous n'avons jamais besoin de manipuler  $\phi$ , mais seulement le noyau  $k$ .

### Exemple 8.1

Dans l'exemple de la figure 8.4, le plongement s'écrit

$$\phi(x_1, x_2) = (x_1, x_2, x_1^2 + x_2^2)$$

Ainsi, avec  $\mathbf{x} = (x_1, x_2)$  et  $\mathbf{x}' = (x'_1, x'_2)$  :

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}') &= \phi(x_1, x_2) \cdot \phi(x'_1, x'_2) \\ &= x_1 x'_1 + x_2 x'_2 + (x_1^2 + x_2^2)(x_1'^2 + x_2'^2) \\ &= \mathbf{x} \cdot \mathbf{x}' + \|\mathbf{x}\|^2 \|\mathbf{x}'\|^2 \end{aligned}$$

Nous avons explicité le plongement  $\phi$  pour des raisons pédagogiques, mais en règle générale on construit un noyau adapté à la nature des données, et le plongement  $\phi$  dans

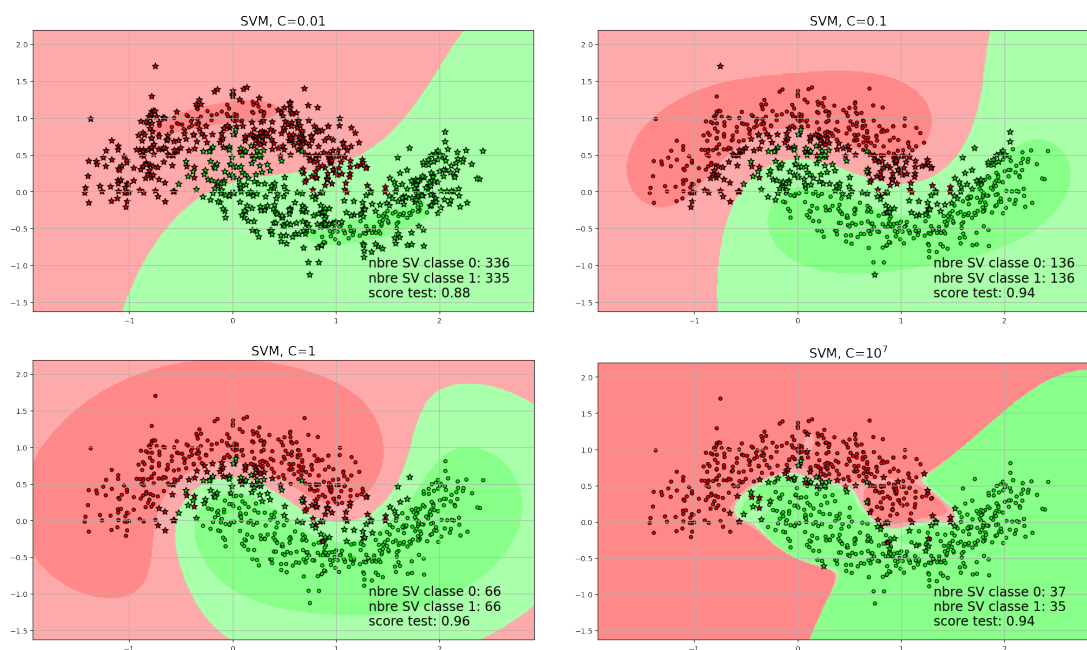


FIGURE 8.5 – Exemples de classification obtenues dans un problème binaire (classe 0 en rouge et classe 1 en vert) pour quatre valeurs de  $C$ . Les vecteurs supports sont marqués d'une étoile. On indique à chaque fois le nombre de vecteurs supports dans chaque classe ainsi que le score de classification obtenues sur des observations test (non représentées) qui n'ont pas servi à l'apprentissage. La marge apparaît en clair. Cette expérience illustre qu'un  $C$  petit induit un sous-apprentissage, et un  $C$  trop grand un sur-apprentissage. Figure à voir en couleur.

L'espace vectoriel de redescription  $\mathcal{H}$  n'est pas explicité (souvent ce n'est d'ailleurs pas facile, et pas informatif) : tout ce dont nous avons besoin est le noyau  $k$ .

### Exemple 8.2

La figure 8.5 montre des exemples de classifications obtenues avec un noyau RBF (voir la section suivante) pour différentes valeurs de l'hyperparamètre  $C$ .

### 8.3.3 Ingénierie des noyaux

S'il est vrai qu'on n'a pas besoin de manipuler le plongement  $\phi$  de  $\mathbb{R}^d$  vers l'espace de redescription  $\mathcal{H}$  mais uniquement le noyau  $k$ , on ne peut tout de même pas choisir n'importe quelle fonction comme noyau :  $k$  doit représenter un produit scalaire dans un espace  $\mathcal{H}$  de manière à ce que la SVM cherchée soit un séparateur linéaire dans  $\mathcal{H}$ .

Sans rentrer dans les détails, un noyau vérifiant les conditions suivantes, dites conditions de Mercer, est valide (c'est-à-dire qu'il permet la représentation d'un produit scalaire dans un espace de redescription  $\mathcal{H}$ ).

**Proposition 8.2** Soit une fonction  $k$  définie sur  $\mathbb{R}^d \times \mathbb{R}^d$  à valeur dans  $\mathbb{R}$ . Si  $k$  est continue symétrique et est telle que pour tout  $n$  et tout ensemble de vecteurs  $\mathbf{x}_1, \dots, \mathbf{x}_n$  de  $\mathbb{R}^d$ , la matrice symétrique  $(k(\mathbf{x}_i, \mathbf{x}_j))_{1 \leq i, j \leq n}$  est positive, alors  $k$  est un noyau valide.

*Rappel* (voir annexe A.3) : la matrice carrée d'ordre  $n$   $(k(\mathbf{x}_i, \mathbf{x}_j))_{1 \leq i, j \leq n}$  est symétrique positive si pour tout  $n$ -uplet  $(c_1, \dots, c_n)$ ,  $\sum_{i, j} k(\mathbf{x}_i, \mathbf{x}_j) c_i c_j \geq 0$ .

*Remarque.* Si  $k$  est un noyau valide, alors le terme  $\sum_{n=1}^N \sum_{m=1}^N \lambda_n \lambda_m y_n y_m k(\mathbf{x}_n, \mathbf{x}_m)$  apparaissant dans le problème (*dual 3*) est positif ou nul, et est une fonction convexe des  $\lambda_n$ . La fonction objectif du problème (*dual 3*) est donc concave et la solution du problème de maximisation est unique. Voir la caractérisation de la convexité pour les fonctions de classe  $C^2$  en annexe B.1.

Par exemple, lorsque  $k(\mathbf{x}, \mathbf{y}) = \mathbf{x} \cdot \mathbf{y}$  (produit scalaire canonique),  $k$  vérifie bien les conditions de Mercer car la matrice des  $k(\mathbf{x}_n, \mathbf{x}_m)$  est une matrice de Gram. Dans ce cas, le plongement  $\phi$  associé est tout simplement l'identité. Il s'agit du noyau linéaire, et on parle dans ce cas de SVM linéaire. Bien entendu, il s'agit tout simplement de la solution des problèmes (*primal 2*)/(*dual 2*) introduits précédemment.

Le choix du noyau doit être adapté au problème considéré; on rencontre l'expression « ingénierie des noyaux » (*kernel design / kernel engineering*).

Il est facile d'utiliser les conditions de Mercer pour démontrer la proposition suivante, qui permet de construire de nouveaux noyaux à partir de noyaux connus.

**Proposition 8.3** Supposons que  $k_1$  et  $k_2$  sont des noyaux valides,  $c > 0$ ,  $A$  une matrice symétrique positive,  $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$  où  $\mathbf{x}_a$  et  $\mathbf{x}_b$  sont des variables sur  $E_a$  et  $E_b$  et  $k_a$  et  $k_b$  des noyaux valides sur ces espaces. Alors :

- $k(\mathbf{x}, \mathbf{y}) = ck_1(\mathbf{x}, \mathbf{y}) + k_2(\mathbf{x}, \mathbf{y})$
- $k(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T A \mathbf{y}$
- $k(\mathbf{x}, \mathbf{y}) = k_1(\mathbf{x}, \mathbf{y}) k_2(\mathbf{x}, \mathbf{y})$
- $k(\mathbf{x}, \mathbf{y}) = k_a(\mathbf{x}_a, \mathbf{y}_a) + k_b(\mathbf{x}_b, \mathbf{y}_b)$
- $k(\mathbf{x}, \mathbf{y}) = k_a(\mathbf{x}_a, \mathbf{y}_a) k_b(\mathbf{x}_b, \mathbf{y}_b)$

définissent des noyaux valides.

Il y a un certain nombre d'autres recettes du même type, voir la littérature sur le sujet.

Par exemple, un noyau populaire sur  $\mathbb{R}^d \times \mathbb{R}^d$  est le noyau polynomial :

$$k(\mathbf{x}, \mathbf{y}) = \gamma(\mathbf{x} \cdot \mathbf{y} + \rho)^\delta$$

où  $\gamma > 0$ ,  $\rho > 0$ ,  $\delta$  entier  $\geq 1$  sont des paramètres.

**Proposition 8.4** Le noyau polynomial est valide.



*Démonstration*

Ce noyau est valide d'après la proposition 8.3. Il est possible de démontrer que l'espace de redescription a pour dimension  $\binom{d+\delta}{\delta}$ .  $\square$

Un autre noyau populaire sur  $\mathbb{R}^d \times \mathbb{R}^d$  est le noyau RBF (*Radial basis function*, ou noyau gaussien) :

$$k(\mathbf{x}, \mathbf{y}) = \exp(-\gamma \|\mathbf{x} - \mathbf{y}\|_2^2)$$

où  $\gamma > 0$  est un paramètre.

Il s'agit d'un noyau dit invariant par translation car il s'écrit sous la forme  $h(\mathbf{x} - \mathbf{y})$  et, vérifie donc la relation  $\forall \mathbf{t} \in \mathbb{R}^d, \forall (\mathbf{x}, \mathbf{y}) \in \mathbb{R}^d \times \mathbb{R}^d, k(\mathbf{x} + \mathbf{t}, \mathbf{y} + \mathbf{t}) = k(\mathbf{x}, \mathbf{y})$ .

**Proposition 8.5** *Le noyau RBF est valide**Démonstration*

Rappelons tout d'abord que si la transformée de Fourier  $\hat{h}$  d'une fonction intégrable  $h$  est définie par l'expression :

$$\hat{h}(\boldsymbol{\xi}) = \int h(\mathbf{x}) e^{-i\mathbf{x} \cdot \boldsymbol{\xi}} d\mathbf{x}$$

alors la transformée de Fourier inverse s'écrit :

$$h(\mathbf{x}) = \frac{1}{2\pi} \int \hat{h}(\boldsymbol{\xi}) e^{i\mathbf{x} \cdot \boldsymbol{\xi}} d\boldsymbol{\xi}$$

On vérifie à présent la condition de Mercer.

Pour tout  $n$ -uplet d'observations  $(\mathbf{x}_1, \dots, \mathbf{x}_n)$  et  $n$ -uplet de scalaires  $(c_1, \dots, c_n)$ ,

$$\begin{aligned} \sum c_j c_k h(\mathbf{x}_j - \mathbf{x}_k) &= \frac{1}{2\pi} \sum_{j,k} c_j c_k \int \hat{h}(\boldsymbol{\xi}) e^{i(\mathbf{x}_j - \mathbf{x}_k) \cdot \boldsymbol{\xi}} d\boldsymbol{\xi} \\ &= \frac{1}{2\pi} \int \hat{h}(\boldsymbol{\xi}) \left( \sum_{j,k} c_j e^{i\mathbf{x}_j \cdot \boldsymbol{\xi}} c_k e^{-i\mathbf{x}_k \cdot \boldsymbol{\xi}} \right) d\boldsymbol{\xi} \\ &= \frac{1}{2\pi} \int \hat{h}(\boldsymbol{\xi}) \left| \sum_j c_j e^{i\mathbf{x}_j \cdot \boldsymbol{\xi}} \right|^2 d\boldsymbol{\xi} \end{aligned}$$

qui est positif lorsque  $\hat{h}$  est une fonction réelle positive. C'est bien le cas pour une fonction  $h$  gaussienne car la transformée de Fourier d'une gaussienne est une gaussienne.  $\square$

Selon cette démonstration, on peut généraliser et dire que toute fonction continue invariante par translation dont la transformée de Fourier est une fonction réelle positive définit un noyau valide.

On peut démontrer que l'espace de redescription associé au noyau RBF est un espace de Hilbert, une sorte d'espace vectoriel de dimension infinie, muni d'un produit scalaire.

Les noyaux précédents sont définis sur des observations vectorielles. Il se trouve que l'on peut également définir des noyaux sur des chaînes de caractères, des ensembles, ou même des graphes. Les noyaux sur les chaînes de caractères sont particulièrement intéressants dans le cadre de la bioinformatique. Par exemple, les protéines sont décrites par des séquences d'acides aminés. Or les protéines de tous les êtres vivants connus ne sont constituées que de 22 acides aminés, représentés par une lettre de l'alphabet : prédire des propriétés de protéine à l'aide de machines à noyau revient à utiliser des noyaux sur des chaînes de 22 caractères.

## 8.4 Retour sur les classifieurs du cours

On peut légitimement se demander ce qui différencie les classifieurs vus dans ce cours, beaucoup d'entre eux sont d'ailleurs linéaires. Nous donnons des éléments de réponse en les présentant dans un cadre unifié.

### 8.4.1 Problèmes d'optimisation

Quasiment tous les classifieurs sont des solutions de problèmes d'optimisation : l'apprentissage supervisé consiste à résoudre des problèmes d'optimisation !

**SVM avec variables d'écart** Comme les écarts  $\xi_n$  vérifient  $\xi_n = \max\{0, 1 - y_n(\mathbf{w} \cdot \mathbf{x}_n + b)\}$ , on peut dire que l'on cherche  $(\mathbf{w}, b)$  minimisant :

$$\|\mathbf{w}\|^2 + C \sum_{n=1}^N \max\{0, 1 - y_n(\mathbf{w} \cdot \mathbf{x}_n + b)\}$$

**Régression logistique** On a vu au chapitre 6 (section 6.2), que l'on cherche  $(\beta_0, \boldsymbol{\beta}_1)$  maximisant la log-vraisemblance conditionnelle

$$\begin{aligned} L(\beta_0, \boldsymbol{\beta}_1) &= \sum_{n=1}^N y_n \log(\sigma(f(\mathbf{x}_n))) + (1 - y_n) \log(\sigma(-f(\mathbf{x}_n))) \\ &= \sum_{n=1}^N \log(\sigma((2y_n - 1)f(\mathbf{x}_n))) \end{aligned}$$

où  $f(\mathbf{x}) = \beta_0 + \boldsymbol{\beta}_1 \cdot \mathbf{x}$ . Dans ces équations, on a utilisé la convention  $y_n \in \{0, 1\}$  utilisée au chapitre 6.

Cela revient, lorsqu'on exprime la relation précédente avec la convention  $y_n \in \{-1, 1\}$  du présent chapitre, à maximiser :

$$L(\beta_0, \boldsymbol{\beta}_1) = \sum_{n=1}^N \log(\sigma(y_n f(\mathbf{x}_n)))$$

soit à minimiser :

$$\sum_{n=1}^N \log(1 + \exp(-y_n(\beta_0 + \boldsymbol{\beta}_1 \cdot \mathbf{x}_n)))$$

La régression logistique régularisée consiste donc à minimiser :

$$\|\beta_1\|_2^2 + C \sum_{n=1}^N \log(1 + \exp(-y_n(\beta_0 + \beta_1 \cdot x_n)))$$

où  $C > 0$ .

**Adaboost** Au chapitre 7 (section 7.3), on a vu qu'on cherchait à minimiser une fonction de coût :

$$E = \sum_{n=1}^N e^{-y_n f(x_n)}$$

où  $f$  est le classifieur obtenu à chaque étape d'Adaboost.

**Perceptron** On anticipe un peu sur le prochain chapitre... L'algorithme du perceptron décrit au chapitre 9 (paragraphe 9.1) cherche à minimiser :

$$\sum_{n=1}^N \max\{0, -y_n(\mathbf{w} \cdot \mathbf{x}_n + b)\}$$

On peut introduire le perceptron régularisé comme solution à la minimisation de :

$$\|\mathbf{w}\|_2^2 + C \sum_{n=1}^N \max\{0, -y_n(\mathbf{w} \cdot \mathbf{x}_n + b)\}$$

où  $C$  est un hyperparamètre positif.

**Fonction de perte** Ces classifieurs Adaboost et les trois classifieurs linéaires SVM, régression logistique, et perceptron) résolvent tous un problème d'optimisation. Ils diffèrent par la manière dont le coût des erreurs sur la base d'apprentissage est intégré. Dans tous les cas, le classifieur  $f$  fait une erreur sur l'observation étiquetée  $(\mathbf{x}_n, y_n)$  lorsque  $z_n = y_n f(\mathbf{x}_n) < 0$ , et le coût d'une erreur est défini par la fonction de perte (*loss function*)  $\ell$  comme suit :

- SVM :  $\ell(z) = \max\{0, 1 - z\}$  (on parle de *hinge loss*, perte « charnière »);
- Régression logistique :  $\ell(z) = \log(1 + \exp(-z))$  (*logistic loss*, perte logistique);
- Adaboost :  $\ell(z) = \exp(-z)$  (*exponential loss*, perte exponentielle);
- Perceptron :  $\ell(z) = \max\{0, -z\}$  (*Perceptron loss*).

### 8.4.2 Comparaison des fonctions de perte (ou coût des erreurs)

La figure 8.6 représente les graphes des différentes fonctions de perte. Ici la perte logistique a été divisée par  $\log(2)$  pour que toutes les fonctions aient la valeur 1 pour  $z = 0$ .

On peut se demander pourquoi on ne cherche pas tout simplement des classifieurs qui minimisent le nombre d'erreurs. Cela reviendrait à considérer la fonction de perte 0-1 (*0-1 loss*) suivante :

$$\ell(z) = \mathbb{1}_{z \leq 0}(z)$$

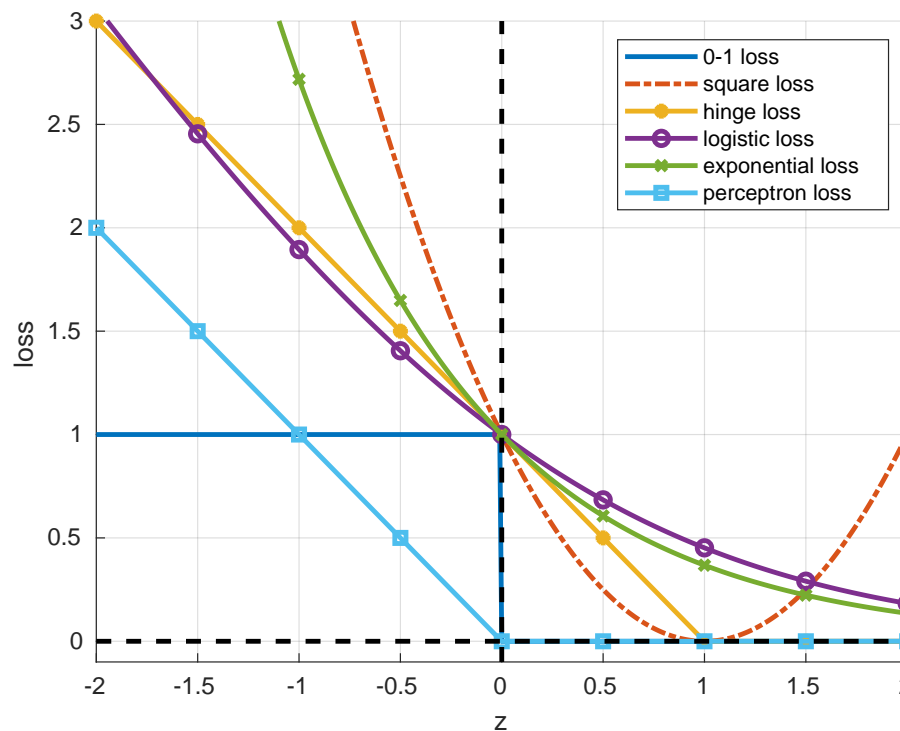


FIGURE 8.6 – Comparaison des fonctions de perte :  $z < 0$  correspond à une observation mal classée,  $z > 0$  à une observation bien classée.

où  $\mathbb{1}_{z \leq 0}(z) = 1$  si  $z \leq 0$  et  $\mathbb{1}_{z \leq 0}(z) = 0$  si  $z > 0$ .

Estimer les paramètres d'un classifieur construit à partir du coût 0-1 nécessite d'utiliser des algorithmes d'optimisation discrète car cette fonction de perte n'est ni différentiable (ni même continue), ni convexe. Les méthodes basées sur le gradient (ou le sous-gradient) ne sont pas utilisables. Le problème d'optimisation est algorithmiquement difficile (comprendre : sa résolution est trop coûteuse en pratique). Les autres fonctions peuvent être vues comme des alternatives convexes au coût « idéal » 0-1, pour lesquelles on dispose d'un cadre théorique et d'algorithmes efficaces.

Par ailleurs, utiliser les autres fonctions peut être plus avantageux que le coût 0-1 : celles-ci pénalisent fortement les observations « vraiment mal classées » ( $z$  très négatif), alors que le coût d'une erreur est toujours 1 pour le coût 0-1, quelle que soit la valeur de  $z$ .

Le coût ou la perte quadratique (*square loss*) s'écrit :

$$\ell(z) = \|1 - z\|_2^2$$

Cette fonction n'est pas adaptée aux problèmes de classification car elle donne une valeur très élevée aux observations trop mal classées ( $z \ll 0$ ) ou trop bien classées ( $z \gg 0$ ) qui deviennent alors des observations influentes (voir la discussion du chapitre 1 et dans un cours de statistique). Ce coût est utilisé dans les problèmes de régression mais dans ce cas il faut

tout de même discuter les observations influentes ou aberrantes, voir le cours de statistique.

Les fonctions de perte de la SVM (*hinge loss*), d'Adaboost (*exponential loss*) et de la régression logistique (*logistic loss*) présentent des graphes similaires, avec une limite nulle lorsque  $z$  tend vers l'infini. Comparée aux SVM, la régression logistique est un peu plus sensible aux observations mal classées et aux observations « trop bien classées » qui sont alors des observations influentes, dans une mesure certes moindre que pour le coût quadratique. Les SVM ne dépendent pas du tout des données bien classées au delà du bord de la marge, car le *hinge loss* est nul pour  $z > 1$ . Adaboost pénalise les observations bien classées de la manière semblable à la régression logistique, mais pénalise bien plus les observations mal classées.

Les classifieurs linéaires fournissent un hyperplan séparateur entre les classes, défini de manière unique dans le cas de la régression logistique ou des SVM. Cet hyperplan diffère d'un modèle à l'autre car il n'est pas solution du même problème d'optimisation. Une SVM présente l'avantage de séparer les deux catégories « au maximum », mais la régression logistique fournit des probabilités a posteriori, ce qui est intéressant pour certains problèmes. Attention néanmoins, ces probabilités sont basées sur une modélisation parfois peu réaliste, comme discuté en section 6.2 du chapitre 6.

### 8.4.3 Quel modèle choisir ?

Les considérations précédentes nous permettent de donner un guide de choix des modèles discutés dans ce cours.

Si la dimension du problème est grande par rapport au nombre d'observations, un classifieur linéaire est possible (car on sait qu'il est « facile » de séparer les classes par un hyperplan en grande dimension) : SVM linéaire ou régression logistique peuvent être envisagées. La SVM linéaire peut être préférée à la régression logistique pour les raisons précédemment évoquées (dépendance uniquement aux vecteurs supports, qui vérifient  $z < 1$  dans la figure 8.6). Néanmoins les probabilités estimées par la régression logistique peuvent être utiles, notamment en cas d'observations proches de la surface de séparation entre les classes, pour lesquelles on aimerait avoir une indication de confiance. Par ailleurs, on peut tout de même questionner la pertinence d'un problème de classification dans lequel le nombre d'observations est faible par rapport à la dimension, en raison de la malédiction de la dimension (cf. problème de Hughes au paragraphe 2.1.4). Il serait sans doute plus pertinent de réduire la dimension du problème au préalable, plutôt que chercher à classer les observations « brutes ».

Dans le cas où les classes ne sont pas linéairement séparables, une SVM à noyau est à favoriser. Le noyau RBF donne souvent de bons résultats lorsque les observations sont vectorielles. Adaboost avec des troncs de décision comme classifieurs faibles est une méthode assez appréciée en pratique. Bien entendu, on ne sait pas a priori si un problème donné correspond à des classes linéairement séparables.

Dans tous les cas, les réseaux de neurones artificiels du prochain chapitre représentent généralement une solution convenable, mais la convergence de l'apprentissage peut être lente et les hyperparamètres peuvent être délicats à fixer.

## 8.5 Régression à vecteurs supports

Les machines à vecteurs supports permettent aussi de traiter des tâches de régression, par l'intermédiaire de la régression à vecteurs supports (*support vector regression*, SVR).

On suppose disposer d'un ensemble d'observations étiquetées  $(\mathbf{x}_n, y_n)_{1 \leq n \leq N}$  où cette fois  $y_n \in \mathbb{R}$  : les étiquettes sont à présent réelles.

Pour des raisons pédagogiques, nous allons commencer par chercher à prédire les étiquettes par une relation affine  $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$ . Une solution de ce problème est connue dans le cadre de la régression linéaire, dans laquelle on choisit  $\mathbf{w}$  et  $b$  de manière à minimiser la somme des carrés des écarts des résidus

$$\sum_{n=1}^N (y_n - (\mathbf{w} \cdot \mathbf{x}_n + b))^2$$

Dans le cadre de la régression à vecteurs supports, nous cherchons  $f$  qui induit un écart d'au plus  $\varepsilon$  entre prédictions et étiquettes : les  $(\mathbf{x}_n, y_n)$  doivent être dans un « tube » de rayon  $\varepsilon$  autour des prédictions  $(x, f(\mathbf{x}))$  (appelé  $\varepsilon$ -tube). De plus, nous cherchons la relation affine  $f$  la plus « plate » possible, c'est à dire avec la plus petite valeur de  $\mathbf{w}$ . Cela signifie que dans le cas où les observations  $x_n$  sont réelles, on cherche la droite de régression de pente la plus petite possible satisfaisant la contrainte.

Nous sommes donc amenés à résoudre le problème d'optimisation suivant :

$$\begin{cases} \arg \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|_2^2 \\ \text{sous contraintes : } \forall 1 \leq n \leq N, |y_n - (\mathbf{w} \cdot \mathbf{x}_n + b)| \leq \varepsilon \end{cases}$$

Ce problème n'ayant pas forcément de solution réalisable lorsque  $\varepsilon$  est trop petit, on introduit des écarts  $E_n \geq 0$  tels que :

$$E_n = \begin{cases} 0 & \text{si } |y_n - (\mathbf{w} \cdot \mathbf{x}_n + b)| \leq \varepsilon \\ |y_n - (\mathbf{w} \cdot \mathbf{x}_n + b)| - \varepsilon & \text{sinon} \end{cases}$$

afin de relaxer les contraintes.

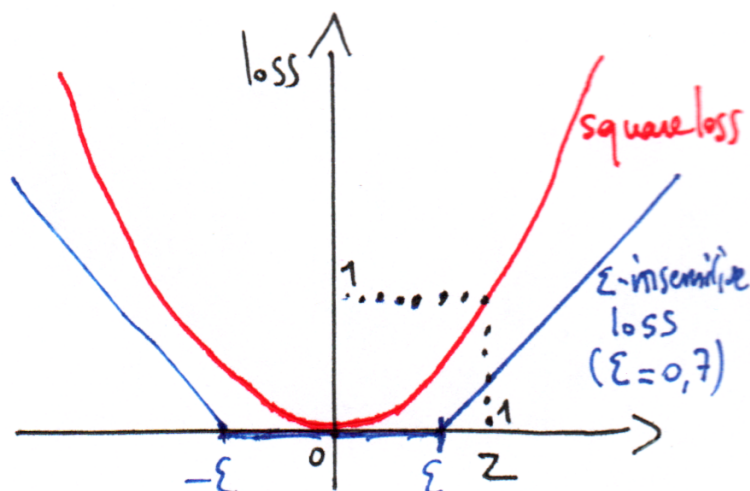
Cela nous conduit à minimiser en fonction de  $\mathbf{w}$ ,  $b$ , et les  $E_n$  la fonction :

$$\frac{1}{2} \|\mathbf{w}\|_2^2 + C \sum_{n=1}^N E_n$$

sous des contraintes du type  $|y_n - (\mathbf{w} \cdot \mathbf{x}_n + b)| - \varepsilon = E_n$ .

Nous sommes donc face à un problème d'optimisation qui ressemble à celui de la régression ridge vue en section 1.4.2.2 (chapitre 1) ; seule la fonction de coût des résidus change. Il s'agit du coût quadratique dans le cas de la régression ridge et d'un coût appelé  $\varepsilon$ -insensitive dans le cas de la SVR. La figure 8.7 illustre les deux fonctions de coût. Ce changement de coût explique la différence fondamentale entre régression ridge et SVR discutée en section 8.6.

Le problème d'optimisation précédent n'est pas quadratique car les contraintes ne sont pas des fonctions linéaires des variables  $\mathbf{w}$ ,  $b$  et  $E_n$ , à cause de la valeur absolue. Pour nous ramener à un tel problème, nous introduisons pour chaque observation des variables d'écart  $\xi_n$  et  $\xi_n^*$  positives et cherchons à résoudre le problème équivalent :

FIGURE 8.7 – Coûts quadratique et  $\epsilon$ -insensitive.

$$\begin{array}{l}
 \text{(primal SVR)} \\
 \text{(\xi}_1, \dots, \xi_n) \text{ et } \xi^* = (\xi_1^*, \dots, \xi_n^*).
 \end{array}
 \left\{ \begin{array}{l}
 \operatorname{argmin}_{\mathbf{w}, b, \xi, \xi^*} \frac{1}{2} \|\mathbf{w}\|_2^2 + C \sum_{n=1}^N (\xi_n + \xi_n^*) \\
 \text{sous contraintes : } \forall 1 \leq n \leq N, y_n - \mathbf{w} \cdot \mathbf{x}_n + b \leq \epsilon + \xi_n \\
 \forall 1 \leq n \leq N, \mathbf{w} \cdot \mathbf{x}_n + b - y_n \leq \epsilon + \xi_n^*
 \end{array} \right. \quad \text{où } \xi =$$

La figure 8.8 montre comment interpréter les variables d'écart. Les vecteurs supports sont les observations  $x_n$  telles que  $(x_n, y_n)$  soit en dehors du  $\epsilon$ -tube.

On démontre alors (exactement comme en section 8.1.2 pour le problème de classification) que le problème dual associé est :

$$\begin{array}{l}
 \text{(dual SVR)} \\
 \text{où } \boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_n) \text{ et } \boldsymbol{\lambda}^* = (\lambda_1^*, \dots, \lambda_n^*).
 \end{array}
 \left\{ \begin{array}{l}
 \operatorname{argmin}_{\boldsymbol{\lambda}, \boldsymbol{\lambda}^*} -\frac{1}{2} \sum_{i,j=1}^N (\lambda_i - \lambda_i^*)(\lambda_j - \lambda_j^*) \mathbf{x}_i \cdot \mathbf{x}_j - \epsilon \sum_{i=1}^N (\lambda_i + \lambda_i^*) + \sum_{i=1}^N y_i (\lambda_i - \lambda_i^*) \\
 \text{sous contraintes : } \forall 1 \leq n \leq N, 0 \leq \lambda_i \leq C \\
 \forall 1 \leq n \leq N, 0 \leq \lambda_i^* \leq C \\
 \sum_{i=1}^N (\lambda_i - \lambda_i^*) = 0
 \end{array} \right.$$

Une fois ce problème dual résolu, on trouve l'expression du régresseur en fonction de  $\boldsymbol{\lambda}$  et  $\boldsymbol{\lambda}^*$  :

$$f(\mathbf{x}) = \sum_{i=1}^N (\lambda_i - \lambda_i^*) \mathbf{x}_i \cdot \mathbf{x} + b$$

où seuls les vecteurs supports contribuent. On trouve également la valeur de  $b$  à partir des vecteurs supports.

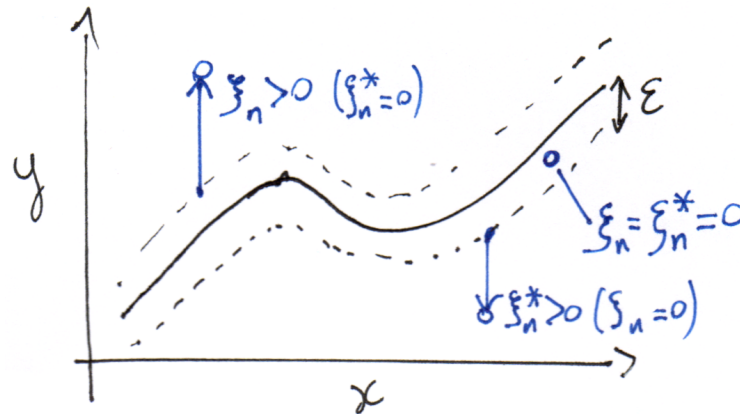


FIGURE 8.8 – Valeurs possibles pour les variables d'écart selon la position relative des observations et de la courbe de régression optimale. Cette représentation anticipe un peu la discussion et l'introduction du noyau pour rendre la prédiction non-linéaire. Trois cas sont possibles à l'optimum du problème de minimisation sous-contrainte définissant la SVR : soit l'observation est dans le  $\varepsilon$ -tunnel et dans ce cas  $\xi_n = \xi_n^* = 0$ , soit l'observation est « au-dessus » et  $\xi_n > 0$ ,  $\xi_n^* = 0$ , soit l'observation est « en-dessous » et  $\xi_n^* > 0$ ,  $\xi_n = 0$ .

Le problème dual ne faisant intervenir les observations que par des produits scalaires, il est possible d'utiliser l'astuce du noyau pour traiter des problèmes de régression non-linéaire. Le régresseur obtenu sera de la forme :

$$f(\mathbf{x}) = \sum_{i=1}^N (\lambda_i - \lambda_i^*) k(\mathbf{x}_i, \mathbf{x}) + b$$

L'approche de la régression expliquée dans cette section, conduisant au régresseur  $f$  ci-dessus, est la *support vector regression*. Deux hyper-paramètres doivent être précisés : le rayon  $\varepsilon$  du « tube » dans lequel les observations étiquetées doivent se trouver pour ne pas pénaliser la régression, et le poids  $C$  des variables d'écart. La figure 8.9 montre un exemple de régression à vecteurs supports sur des observations réelles. Cette approche permet de trouver des modèles réguliers même lorsque les données d'apprentissage sont bruitées ou aberrantes.

Tous les détails peuvent être trouvés dans : A.J. Smola, B. Schölkopf, *A tutorial on support vector regression*, *Statistics and Computing* 14 :199-222, 2004.

## 8.6 Pour approfondir...

**Lecture** Pour un tutoriel complet sur les SVM, se référer à :

C. Burges, *A tutorial on support vector machines for pattern recognition*, *Data Mining and Knowledge Discovery*, vol. 2, no. 2, p. 121-167, 1998.



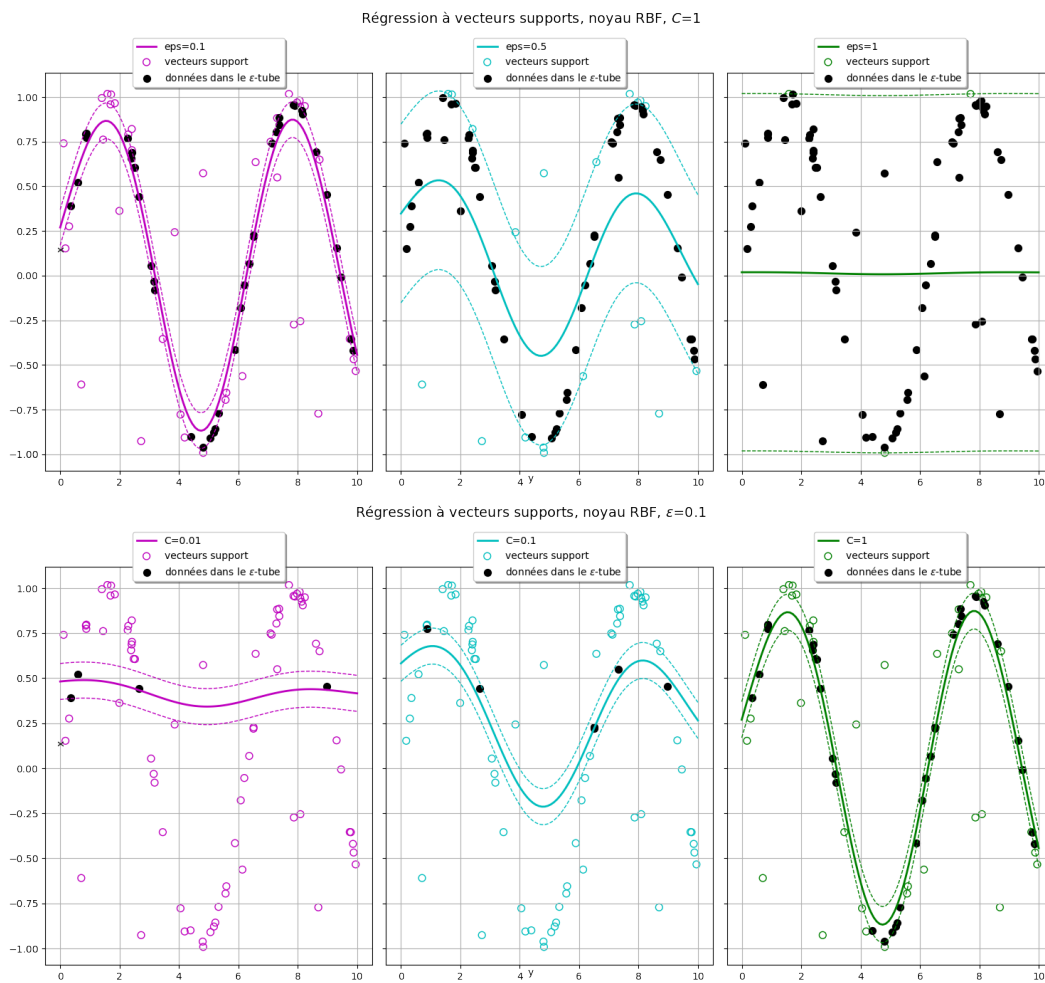


FIGURE 8.9 – *Régression à vecteurs supports, noyau RBF.* Parmi les observations d'apprentissage, les vecteurs supports sont indiqués comme des cercle, les observations dans le  $\epsilon$ -tube (représenté par les lignes pointillées) comme des disques noirs. Rappelons que seuls les vecteurs supports interviennent dans la définition du régresseur. Première ligne :  $C = 1$ , et de gauche à droite  $\epsilon = 0.1, 0.5$ , et  $1$ . Plus  $\epsilon$  est grand, moins il y a de vecteurs supports. Seconde ligne :  $\epsilon = 0.1$ , et de gauche à droite  $C = 0.01, 0.1$ , et  $1$ . À  $\epsilon$  fixé, plus  $C$  est grand plus les variables d'écart sont pénalisées.

**No-free-lunch theorem en apprentissage supervisé** Dans la comparaison des différents modèles de classification, il ne faut pas perdre de vue le *No-free-lunch theorem* de l'apprentissage automatique : aucun algorithme n'aura de performances meilleures en moyenne sur tous les problèmes. Le nom du théorème vient de l'expression *There ain't no such thing as a free lunch* et du scénario suivant : chaque restaurant (classifieur) propose un menu fait de plats (des problèmes de classification), chaque plat étant associé à un prix (le coût global d'erreur). Les

menus des restaurants sont les mêmes, seuls les prix des plats sont permutés (les classifieurs ont des performances différentes sur les problèmes). Pour un client choisissant ses plats au hasard, il n'y a pas de restaurant meilleur marché qu'un autre, le coût moyen est constant. Il n'y a pas de choix de restaurant conduisant à un menu gratuit. Autrement dit, il n'y a pas de classifieur ayant des performances meilleures en moyenne sur tous les problèmes.

Une présentation rigoureuse du *no-free-lunch theorem* est disponible dans l'article : D.H. Wolpert, *The Supervised Learning No-Free-Lunch Theorems*, in *Soft Computing and Industry: Recent Applications*, p. 25-42, 2002.

Un article de vulgarisation récent montre la difficulté de la recherche du « meilleur » modèle d'apprentissage et relativise les progrès de certains domaines applicatifs :

M. Hutson, *Eye-catching advances in some AI fields are not real*, in *Science News*, May 27, 2020. <https://www.science.org/content/article/eye-catching-advances-some-ai-fields-are-not-real>

**Régression ridge à noyau** La régression ridge (section 1.4.2.2 du chapitre 1) définit la fonction  $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$  (en adoptant les notations du présent chapitre), de telle manière que  $\mathbf{w}$  minimise :

$$\sum_{n=1}^N |y_n - b - \mathbf{w} \cdot \mathbf{x}_n|^2 + \lambda \|\mathbf{w}\|_2^2$$

où  $\lambda$  est un paramètre positif.

En remplaçant le produit scalaire par un noyau de Mercer  $k$ , on va définir la régression ridge à noyau (*kernel ridge regression*). Notons  $\phi$  le plongement tel que  $k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{y})$ . La fonction  $f(\mathbf{x}) = b + \mathbf{w} \cdot \phi(\mathbf{x})$  cherchée est telle que  $\mathbf{w}$  minimise :

$$\sum_{n=1}^N |y_n - b - \mathbf{w} \cdot \phi(\mathbf{x}_n)|^2 + \lambda \|\mathbf{w}\|_2^2$$

Il se trouve que le développement suivant ne se déroule pas bien si  $b \neq 0$ <sup>1</sup>. Néanmoins, la section 1.4.2.2 suggère de centrer les données au préalable. Si les  $y_n$  sont de moyenne nulle, nous pouvons retirer  $b$  du modèle. Reprenons alors les notations des sections 1.4.2.1 (mais sans intégrer la colonne de 1 dans  $X$  puisque l'ordonnée à l'origine n'intervient plus) et 1.4.2.2 :  $W$  vérifie

$$W = (X^T X + \lambda I_d)^{-1} X^T Y$$

Grâce au *matrix inversion lemma* aussi appelé identité de Woodbury (demandez à Google) :

$$(X^T X + \lambda I_d)^{-1} X^T = \lambda X^T (I_N - X X^T)^{-1}$$

et donc :

$$W = X^T (X X^T + \lambda I_N)^{-1} Y$$

Ici,  $X X^T$  est la matrice carrée d'ordre  $N$  dont l'élément en ligne  $i$  colonne  $j$  est  $\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j)$ . Notons  $K = X X^T$  (remarquons que le raisonnement est ici un peu heuristique : si

1. Si  $b \neq 0$ , il faut remplacer  $\lambda I_d$  par  $\Lambda$  (paragraphe 1.4.2.2 du chapitre 1) dans ce qui suit et l'identité de Woodbury ne peut pas être utilisée car  $\Lambda$  n'est pas inversible

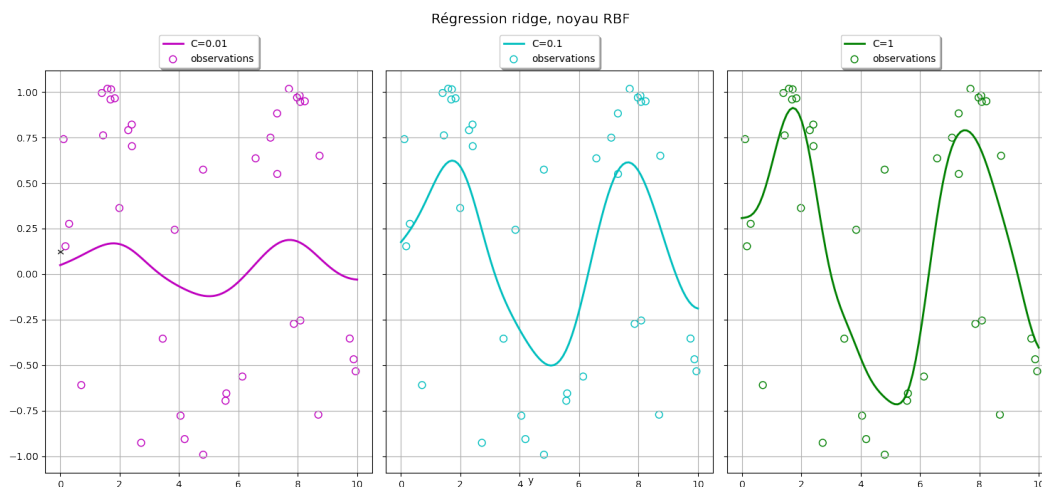


FIGURE 8.10 – Régression ridge à noyau RBF Plus  $C$  est grand, plus les écarts sont pénalisés et plus la courbe de régression s'approche des observations. Un  $C$  trop faible implique un sous-apprentissage, un  $C$  trop élevé un sur-apprentissage. À comparer à la figure 8.9.

les observations sont plongées dans un espace de Hilbert de dimension infinie, les matrices  $W$  et  $X$  ne sont pas définies; l'important est que  $K$  le soit bien, ce qui est le cas car le nombre d'observations  $N$  est fini). On a alors  $W = X^T(K + \lambda I_N)^{-1}Y$ . Notons que  $K$  est symétrique.

Réintégrons la moyenne  $b$  des  $y_n$ . Face à une nouvelle observation  $\mathbf{x}$ , la fonction  $f$  s'évalue alors comme :

$$f(\mathbf{x}) = b + \mathbf{w} \cdot \phi(\mathbf{x}) = b + W^T \phi(\mathbf{x}) = b + Y^T (K + \lambda I_N)^{-1} X \phi(\mathbf{x})$$

en identifiant  $\phi(\mathbf{x})$  à son vecteur colonne, soit :

$$f(\mathbf{x}) = b + Y^T (K + \lambda I_N)^{-1} \kappa(\mathbf{x})$$

où  $\kappa(\mathbf{x}) = X \phi(\mathbf{x})$  est le vecteur-colonne de composantes  $k(\mathbf{x}_n, \mathbf{x})$ .

Le point essentiel est qu'on a pu plonger les observations dans un espace de grande dimension en faisant uniquement intervenir le noyau  $k$ . L'intérêt est que la fonction  $f$  obtenue n'est plus affine comme pour la régression ridge classique, mais non-linéaire grâce à l'utilisation du noyau  $k$ .

Par comparaison à la régression à vecteurs supports, la régression ridge est donnée par une formule explicite et ne nécessite pas de résoudre un problème d'optimisation. Néanmoins, cette formule nécessite l'inversion de la matrice  $K + \lambda I_N$  d'ordre  $N$ , puis un produit par le vecteur  $\kappa(\mathbf{x})$  de dimension  $N$ , ce qui est difficilement envisageable lorsque le nombre d'observations est grand. Au contraire, le prédicteur final de la SVR ne dépend que des vecteurs de support, qui sont généralement en nombre limité. En effet, dans le cas de la SVR,  $f(\mathbf{x}) = b + \sum_{n=1}^N (\lambda_n - \lambda_n^*) k(\mathbf{x}_n, \mathbf{x})$ , soit  $f(\mathbf{x}) = b + \tilde{\Lambda} \kappa(\mathbf{x})$  où  $\tilde{\Lambda}$  est une matrice diagonale (formée par les  $\lambda_n - \lambda_n^*$ ) dont seuls les termes relatifs aux vecteurs de support sont non-nuls. On

dit que cette matrice  $\tilde{\Lambda}$  est creuse., contrairement à la matrice  $Y^T(K + \lambda I_N)^{-1}$  de la régression ridge à noyau. Cette propriété est une conséquence du fait que la SVR pénalise les résidus avec une norme de type  $\|\cdot\|_1$  qui favorise la parcimonie, alors que la régression ridge utilise la norme euclidienne  $\|\cdot\|_2$  (voir section 1.4.2.1 au chapitre 1).

En réécrivant le problème d'optimisation de la régression logistique sous la forme suivante :

$$\frac{1}{2} \|\mathbf{w}\|_2^2 + C \sum_{n=1}^N |y_n - b - \mathbf{w} \cdot \mathbf{x}_n|^2$$

on fait apparaître un paramètre  $C > 0$  qui joue un rôle équivalent au paramètre  $C$  de la SVR. La figure 8.10 illustre la régression ridge à noyau sur des observations réelles et le rôle de  $C$ .

**Machines à noyau** Des noyaux peuvent être définis pour des observations qui ne sont pas facilement représentables sous forme vectorielle, comme des graphes, des chaînes de caractères... Par ailleurs, l'astuce du noyau est utilisable dès qu'une méthode ne fait apparaître que des produits scalaires. La première utilisation de l'astuce est le *kernel perceptron* en 1964. Une version « à noyau » de l'analyse en composante principale existe (*kernel PCA*), ainsi qu'un *kernel k-means*.

L'article suivant présente plusieurs modèles de « machines à noyaux » ainsi que différents noyaux, pour des données vectorielles ou non :

T. Hofmann, B. Schölkopf, A. Smola, *Kernel methods in machine learning*, Annals of Statistics, vol. 36 no. 3, 1171-1220, 2008.

# Les réseaux de neurones artificiels

Nous discutons à présent les réseaux de neurones artificiels : un modèle qui, au cours des soixante dernières années, s'est incarné sous différentes formes, a suscité des attentes parfois démesurées, est retombé dans l'oubli (relatif), mais a fini par surpasser la concurrence sur certains problèmes au point d'être souvent synonyme d'Intelligence Artificielle dans le discours public.

### 9.1 Le perceptron (neurone artificiel)

Notre histoire commence dans les années 1950 lorsque Frank Rosenblatt, psychologue américain, propose un des premiers algorithmes de l'apprentissage automatique : le perceptron. Ses travaux font suite à la première modélisation du neurone biologique proposée par Walter Pitts et Warren McCulloch en 1943. Rosenblatt propose un algorithme permettant d'adapter les paramètres du modèle d'un neurone en fonction d'exemples d'apprentissage. Voici un extrait de l'introduction de son rapport de recherche de 1957 intitulé "*The Perceptron, a perceiving and recognizing automaton*" :

"Since the advent of electronic computers and modern servo systems, an increasing amount of attention has been focused on the feasibility of constructing a device possessing such human-like functions as perception, recognition, concept formation, and the ability to generalize from experience. In particular, interest has centered on the idea of a machine which would be capable of conceptualizing inputs impinging directly from the physical environment of light, sound, temperature, etc. – the "phenomenal world" with which we are all familiar – rather than requiring the intervention of a human agent to digest and code the necessary information.

A primary requirement of such a system is that it must be able to recognize complex patterns of information which are phenomenally similar, or are experimentally related – a process which corresponds to the psychological phenomena of "association" and "stimulus generalization". The system must recognize the "same" object in different orientations, sizes, colors, or transformations, and against a variety of different backgrounds."

En langage moderne, l'objectif est bien de résoudre des problèmes de classification super-

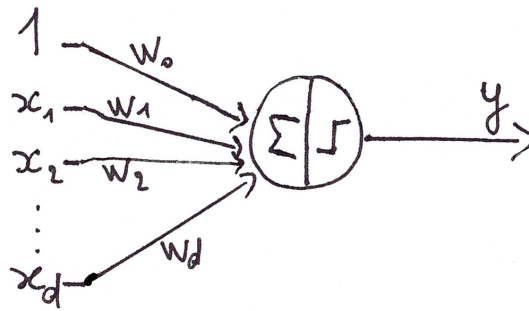


FIGURE 9.1 – *Modèle de perceptron de Rosenblatt* :  $y = 1$  si  $w_0 + \sum_{i=1}^d w_i x_i > 0$  ;  $y = -1$  sinon.

visée à partir d'observations qui seraient des images ou des sons. Une telle présentation du perceptron est très optimiste, mais, par certains aspects, les réseaux de neurones convolutifs apparus il y a quelques années commencent à atteindre cet objectif.

Nous allons présenter le perceptron comme un classifieur biclasse. Comme l'illustre la figure 9.1, le perceptron admet  $d$  valeurs scalaires  $(x^1, x^2, \dots, x^d)$  en entrée, et retourne en sortie 1 ou -1 selon le signe de  $w_0 + w_1 x^1 + \dots + w_d x^d$ , où  $w_0, \dots, w_d$  sont les paramètres du modèle. En langage moderne, le perceptron calcule de quel côté d'un hyperplan affine se situe une observation  $\mathbf{x} = (x^1, \dots, x^d)$ . Il s'agit donc d'un classifieur linéaire. Les  $d+1$  poids  $w_i$  doivent être adaptés selon les exemples d'apprentissage.

Même si sa représentation graphique évoque le neurone biologique (ou en tout cas un modèle assez grossier de neurone), le perceptron se contente juste de mettre en œuvre un classifieur linéaire.

### 9.1.1 Algorithme d'apprentissage du perceptron

F. Rosenblatt a proposé un algorithme permettant de fixer les paramètres de son perceptron en fonction d'une base d'exemples (ou d'apprentissage).

La base d'exemples est formée de  $N$  observations étiquetées  $(\mathbf{x}_n, y_n)_{1 \leq n \leq N}$ , où pour tout  $n$ ,  $\mathbf{x}_n \in \mathbb{R}^d$  et  $y_n \in \{-1, 1\}$ .

Quitte à considérer  $\mathbf{x}_n \in \mathbb{R}^{d+1}$  avec une première composante  $x^0$  égale à 1, on peut toujours se ramener à des hyperplans de  $\mathbb{R}^{d+1}$  d'équation  $\mathbf{w} \cdot \mathbf{x}$ . On adoptera cette convention pour alléger les notations.

L'algorithme d'apprentissage du perceptron, permettant d'adapter les poids  $\mathbf{w}$  en fonction de la base d'exemples  $(\mathbf{x}_n, y_n)_{1 \leq n \leq N}$ , est le suivant :

- Initialisation :  $\mathbf{w} = 0$
- Tant que la valeur de  $\mathbf{w}$  est mise à jour lors d'un parcours de la base d'exemples, répéter :
  - Pour  $n$  de 1 à  $N$  :
    - calculer  $\hat{y}_n = \text{signe}(\mathbf{w} \cdot \mathbf{x}_n)$
    - si  $\hat{y}_n \neq y_n$  alors mettre à jour  $\mathbf{w}$  :  $\mathbf{w} \leftarrow \mathbf{w} + y_n \mathbf{x}_n$
- Retourner  $\mathbf{w}$

Ici,  $\text{signe}(z) = -1$  si  $z < 0$ , et  $\text{signe}(z) = 1$  sinon.

À chaque étape, l'algorithme compare la « vraie étiquette »  $y_n$  et l'étiquette prédite  $\hat{y}_n$  par le perceptron sur une observation  $\mathbf{x}_n$ , cette étiquette étant calculée à l'aide de l'estimation courante des poids  $\mathbf{w}$ . Si  $\hat{y}_n$  ne correspond pas à  $y_n$ , les poids du perceptron sont mis à jour. L'algorithme est donc *error-driven* (les mises à jour visent à corriger les erreurs), et *online* (les exemples sont présentés successivement).

**Théorème 9.1** (Novikoff 1962) *Si les observations étiquetées sont linéairement séparables, alors l'algorithme du perceptron aboutit en un nombre fini d'étapes aux paramètres d'un hyperplan séparateur.*

#### Démonstration

Soit  $\mathbf{w}$  définissant un hyperplan séparateur des deux classes (qui existe par hypothèse). Les poids du perceptron correspondant sont définis à une constante multiplicative près (seul le signe de  $\mathbf{w} \cdot \mathbf{x}_n$  importe), on suppose donc sans perte de généralité  $\|\mathbf{w}\|_2 = 1$ .

Chaque hyperplan séparateur définit une marge entre les deux classes comme la plus petite distance entre les observations de chaque classe et l'hyperplan. Rappelons que la distance entre une observation  $\mathbf{x}$  et l'hyperplan défini par  $\mathbf{w}$  est  $|\mathbf{w} \cdot \mathbf{x}| / \|\mathbf{w}\|_2$ . La marge  $\gamma$  est donc  $\gamma = \min_{1 \leq n \leq N} y_n \mathbf{w} \cdot \mathbf{x}_n$  car on suppose  $\|\mathbf{w}\|_2 = 1$ .

Notons également  $R = \max_{1 \leq n \leq N} \|\mathbf{x}_n\|_2$ .

Considérons la  $k$ -ème erreur de l'algorithme, commise sur l'observation étiquetée  $(\mathbf{x}_n, y_n)$ . On a :  $y_n \mathbf{w}_k \cdot \mathbf{x}_n < 0$  (car  $y_n$  et  $\mathbf{w}_k \cdot \mathbf{x}_n$  ont des signes différentes), et la mise à jour s'écrit  $\mathbf{w}_{k+1} = \mathbf{w}_k + y_n \mathbf{x}_n$ .

D'une part :  $\mathbf{w}_{k+1} \cdot \mathbf{w} = \mathbf{w}_k \cdot \mathbf{w} + y_n \mathbf{x}_n \cdot \mathbf{w} \geq \mathbf{w}_k \cdot \mathbf{w} + \gamma$  par définition de  $\gamma$ .

Donc par récurrence :  $\mathbf{w}_{k+1} \cdot \mathbf{w} \geq k\gamma$  car  $\mathbf{w}_0 = 0$ .

D'autre part :  $\|\mathbf{w}_{k+1}\|_2^2 = \|\mathbf{w}_k + y_n \mathbf{x}_n\|_2^2 = \|\mathbf{w}_k\|_2^2 + 2y_n \mathbf{x}_n \cdot \mathbf{w}_k + \|\mathbf{x}_n\|_2^2 \leq \|\mathbf{w}_k\|_2^2 + R^2$  car  $y_n \mathbf{w}_k \cdot \mathbf{x}_n < 0$  et  $\|\mathbf{x}_n\|_2^2 \leq R^2$ .

Donc par récurrence :  $\|\mathbf{w}_{k+1}\|_2^2 \leq kR^2$ .

Comme  $\|\mathbf{w}\|_2 = 1$ ,  $\mathbf{w}_{k+1} \cdot \mathbf{w} / \|\mathbf{w}_{k+1}\|_2$  est le cosinus de l'angle formé par les vecteurs  $\mathbf{w}_{k+1}$  et  $\mathbf{w}$ . Nous déduisons des deux inégalités précédentes, pour tout  $k$  :

$$\sqrt{k}\gamma/R \leq \mathbf{w}_{k+1} \cdot \mathbf{w} / \|\mathbf{w}_{k+1}\|_2 \leq 1$$

Autrement dit, au cours des mises à jour l'hyperplan défini par le perceptron (paramètre  $\mathbf{w}_k$ ) tend à s'aligner sur l'hyperplan séparateur de paramètre  $\mathbf{w}$ , car le cosinus de leur angle se

rapproche de 1.

Par ailleurs,  $\sqrt{k}\gamma/R \leq 1$ , donc  $k \leq R^2/\gamma^2$  : cela signifie que le nombre de mises à jour est au plus  $R^2/\gamma^2$  avant que le perceptron ne fasse plus d'erreurs.  $\square$

La démonstration nous indique que plus la marge  $\gamma$  entre les deux classes à discriminer est grande (relativement à  $R$ ), plus la convergence est rapide. Le nombre de mises à jour est indépendant de la dimension  $d$ , ainsi que du nombre d'observations  $N$ . Cela ne signifie pas que la vitesse de convergence est indépendante de  $N$ , car plus  $N$  est grand plus on peut mettre de temps à « tomber » sur les erreurs de classification permettant la mise à jour.

Si les données ne sont pas séparables, l'algorithme ne converge pas. Il faut alors arrêter après un certain nombre de parcours des données. Un parcours de la base d'apprentissage en entier est appelé époque, ou *epoch*.

L'algorithme d'apprentissage du perceptron nous fournit un des hyperplans séparateurs, de manière relativement arbitraire (cela dépend de l'ordre de parcours des observations). Comme l'illustre la figure 9.2, l'indétermination pose problème pour la prédiction de la classe de nouvelles observations.

L'algorithme du perceptron peut être vu comme la descente de gradient stochastique minimisant le coût d'erreur

$$\ell(\mathbf{w}) = \sum_{n=1}^N \max\{0, -y_n \mathbf{w} \cdot \mathbf{x}_n\}$$

(voir l'annexe B.3, en particulier la section B.3.3 pour le gradient stochastique, ainsi que la section 8.4.1 pour la discussion des fonctions de perte).

En effet, si  $y_n \mathbf{w} \cdot \mathbf{x}_n > 0$  (prédiction correcte) alors  $\ell(\mathbf{w}) = 0$  sur un voisinage de  $\mathbf{w}$  donc  $\nabla \ell(\mathbf{w}) = 0$ , et si  $y_n \mathbf{w} \cdot \mathbf{x}_n < 0$  (prédiction incorrecte) alors  $\ell(\mathbf{w}) = -y_n \mathbf{w} \cdot \mathbf{x}_n$  donc  $\nabla \ell(\mathbf{w}) = -y_n \mathbf{x}_n$ . L'algorithme de descente de gradient à pas  $\eta = 1$  (voir annexe B.3) correspond bien à l'algorithme d'apprentissage du perceptron.

Le théorème 9.1 nous assure la convergence dans le cas où les classes sont séparables, et seulement dans ce cas.

### 9.1.2 Perceptron multiclasse

Le perceptron peut être étendu au cas de  $K > 2$  classes, avec des étiquettes  $y \in \{1, 2, \dots, K\}$ . Les classes sont associées à  $K$  vecteurs-poids  $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_K$  (un  $\mathbf{w}_k$  par classe), et la règle de décision est d'associer une observation  $\mathbf{x}$  à l'étiquette  $\hat{y} = \operatorname{argmax}_k \mathbf{w}_k \cdot \mathbf{x}$ . L'apprentissage s'opère par un algorithme similaire au cas biclasse, dans lequel la règle de mise à jour lorsque  $\hat{y}_n \neq y_n$  est :  $\mathbf{w}_{y_n} \leftarrow \mathbf{w}_{y_n} + \mathbf{x}_n$  et  $\mathbf{w}_{\hat{y}_n} \leftarrow \mathbf{w}_{\hat{y}_n} - \mathbf{x}_n$ .

D'autres règles de mise à jour sont possibles dans le cas multiclasse.

### 9.1.3 Limites du perceptron, et comment les dépasser

Si le perceptron « simule » effectivement un neurone, il serait intéressant qu'il puisse faire des calculs logiques. Il se trouve que le perceptron permet de calculer les opérateurs logiques



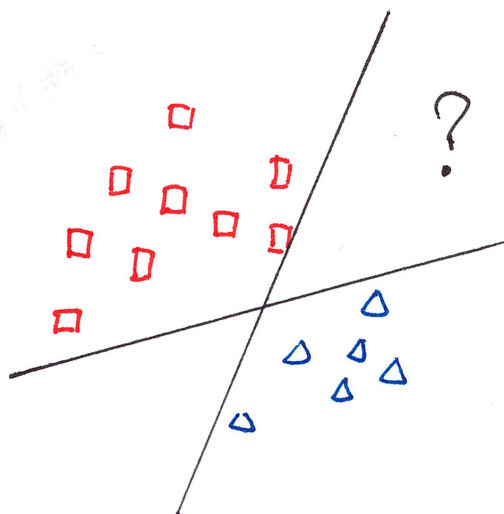


FIGURE 9.2 – Perceptron : ambiguïté de l'hyperplan séparateur obtenu. Selon l'hyperplan retenu, une observation non étiquetée sera associée à une classe ou à l'autre.

AND et OR. Rappelons que, si l'on note 0 le booléen « faux » et 1 le booléen « vrai », les tables de vérité de ces opérateurs sont les suivantes :

| $x_1$ |     | $x_2$ | $y$ |
|-------|-----|-------|-----|
| 0     | AND | 0     | 0   |
| 0     | AND | 1     | 0   |
| 1     | AND | 0     | 0   |
| 1     | AND | 1     | 1   |

| $x_1$ |    | $x_2$ | $y$ |
|-------|----|-------|-----|
| 0     | OR | 0     | 0   |
| 0     | OR | 1     | 1   |
| 1     | OR | 0     | 1   |
| 1     | OR | 1     | 1   |

Comme l'illustre la figure 9.3, le calcul de ces deux opérateurs revient à un problème de classification pour deux classes linéairement séparables. On pourrait penser qu'il s'agit là d'un argument fort pour assimiler le perceptron à une brique essentielle dans la construction d'une « intelligence artificielle ». Cependant, le perceptron ne peut pas calculer l'opérateur logique XOR (« ou exclusif »), qui a pour table de vérité :

| $x_1$ |     | $x_2$ | $y$ |
|-------|-----|-------|-----|
| 0     | XOR | 0     | 0   |
| 0     | XOR | 1     | 1   |
| 1     | XOR | 0     | 1   |
| 1     | XOR | 1     | 0   |

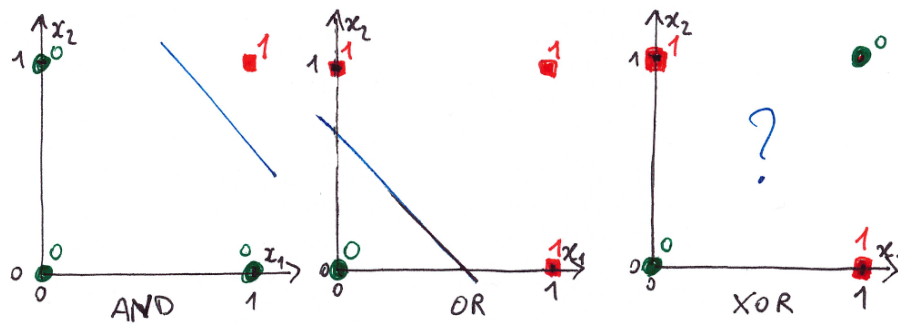


FIGURE 9.3 – Les calculs de AND, OR, ou XOR peuvent être vus comme des problèmes de classification binaire. Les opérateurs logiques AND et OR sont calculés par un classifieur linéaire (donc un perceptron), contrairement à l'opérateur XOR.

En effet, comme l'illustre la figure 9.3, calculer le XOR est équivalent à résoudre un problème de classification non linéaire, ce dont est incapable le perceptron. Ce genre de limitations du perceptron a été la cause de l'abandon des méthodes dites *connexionnistes* en faveur des méthodes dites *symboliques*, cause du premier « hiver de l'IA » dans les années 1970.

Une solution à ce problème était en fait connue, et s'est révélée particulièrement fructueuse par la suite... On remarque la relation suivante entre XOR et OR/AND : quels que soient les booléens  $x_1$  et  $x_2$ ,

$$x_1 \text{ XOR } x_2 = (x_1 \text{ OR } x_2) \text{ AND } (\text{NOT}(x_1) \text{ OR } \text{NOT}(x_2))$$

Pour établir cette relation, on peut calculer la table de vérité de l'expression de droite et constater qu'elle est identique à celle du XOR. La relation précédente permet de construire un réseau de perceptrons à une couche cachée entre la couche d'entrée et la couche de sortie, représenté sur la figure 9.4.

Il y a même mieux : on peut démontrer que toute formule logique (une fonction de  $n$  booléens, à valeurs booléennes) peut être calculée par un réseau de perceptrons à une couche cachée. En effet, les logiciens démontrent le théorème suivant.

**Théorème 9.2** Toute formule logique peut être convertie en une formule équivalente sous forme normale conjonctive.

Une forme normale conjonctive est une conjonction (AND) de disjonction(s) (OR) de littéraux, un littéral étant une variable booléenne ou la négation d'une variable booléenne. La décomposition précédente du XOR est justement une forme normale conjonctive.

On peut alors construire un réseau de perceptrons à une couche cachée calculant une formule logique exprimée sous forme normale conjonctive en appliquant les « recettes » suivantes :

1. on définit la couche d'entrée faites des variables  $x_1, x_2, \dots, x_n$

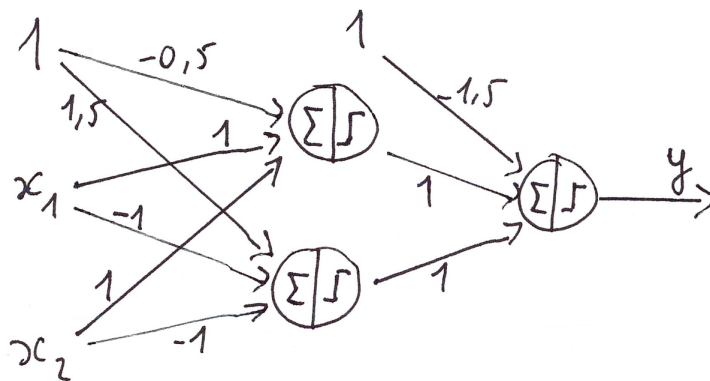


FIGURE 9.4 – Perceptron à une couche cachée calculant le « ou exclusif » (XOR). On vérifie que, selon les valeurs de  $x_1$  et  $x_2$  (0 ou 1), la valeur de  $y$  en sortie correspond bien à  $x_1 \text{ XOR } x_2$ .

2. on définit la couche cachée faite de  $m$  neurones cachés correspondant aux  $m$  disjonctions, ainsi que les poids reliant les neurones cachés à l'entrée, en se basant sur la propriété :

$$a_1 \text{ OR } a_2 \text{ OR } \dots \text{ OR } a_m = H\left(\sum_{i=1}^m a_i - 0.5\right)$$

où on remplace le littéral  $a_i$  par une variable  $x$  ou par  $1 - x$  lorsque  $a_i = \text{NOT}(x_i)$ .

3. on définit les poids liant les  $m$  neurones cachés à la sortie en se basant sur :

$$a_1 \text{ AND } a_2 \text{ AND } \dots \text{ AND } a_m = H\left(\sum_{i=1}^m a_i + 0.5 - m\right)$$

Ici, la fonction d'activation  $H$  est telle que  $H(z) = 0$  si  $z < 0$ , et  $H(z) = 1$  sinon.

Sans rentrer dans les détails, pour convertir une formule logique en forme normale conjonctive, il peut être nécessaire de faire apparaître un grand nombre de disjonctions (plus précisément, une fonction exponentielle du nombre de variables). Cela signifie que le réseau de perceptrons à une couche cachée peut certes calculer toute formule logique, mais parfois au prix d'un grand nombre de neurones dans la couche cachée.

## 9.2 Perceptron multicouche ou réseau de neurones artificiels

Les efforts pour prolonger les travaux de Rosenblatt ont donné naissance au perceptron multicouche (*multilayer perceptron*, MLP), ou réseau de neurones artificiels. La terminologie est légèrement trompeuse : il ne s'agit pas d'un perceptron à plusieurs couches comme celui de la figure 9.4, mais plutôt d'un réseau possédant plusieurs couches de neurones, dont les fonctions d'activation ne seraient pas nécessairement des fonctions en échelon comme pour le perceptron.

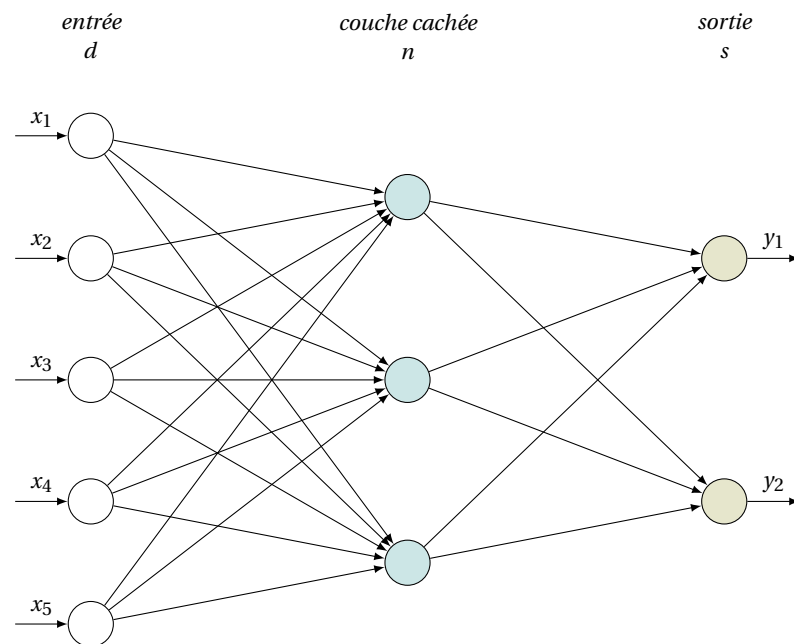


FIGURE 9.5 – Un réseau de neurones artificiels (perceptron multicouche) à une couche cachée. L'« information » en entrée est ici un vecteur de  $\mathbb{R}^5$ , et en sortie un vecteur de  $\mathbb{R}^2$ . Avec  $d$  neurones en entrée,  $n$  neurones dans la couche cachée, et  $s$  neurones en sortie, le nombre de paramètres (poids et biais) dans ce modèle est  $(d+1)n + (n+1)s$  (« +1 » à cause du biais en chaque neurone).

La figure 9.5 montre la représentation graphique d'un perceptron multicouche. Un tel réseau ne présente pas de boucle ou cycle, l'information se propage de la couche d'entrée à la couche de sortie : on parle de réseau à propagation avant (*feedforward neural network*). Les couches de neurones situés entre l'entrée et la sortie sont appelées couches cachées. Chaque couche est un ensemble de neurones n'ayant pas de connexions entre eux, et correspond à un « niveau » dans le vocabulaire de la théorie des graphes.

Un perceptron multicouche traite des vecteurs de  $\mathbb{R}^d$  : la couche d'entrée est composée de  $d$  neurones qui lisent les composantes d'un vecteur et envoient l'information aux neurones de la première couche cachée. Chaque neurone d'une couche cachée fait une moyenne pondérée des informations reçues et ré-émet aux neurones de la couche suivante cette information moyenne modifiée par une fonction d'activation.

Plusieurs couches cachées peuvent se succéder, chacune étant constituée d'un certain nombre de neurones connectés aux neurones de la couche précédente. L'« information » arrivant à un des neurones ( $j$ ) de la couche  $k$  est

$$a_j = \sum_i w_{ji} z_i + b_j$$

où :

- la somme porte sur l'ensemble des neurones  $i$  de la couche précédente  $k - 1$  connectés avec le neurone  $j$ ,
- $w_{ji}$  est le poids de la connexion entre le  $i$ -ème neurone de la couche  $k - 1$  et le neurone  $j$ ,
- $b_j$  (appelé « biais ») est un scalaire,
- et  $z_i$  est le signal émis par ce  $i$ -ème neurone.

Le neurone  $j$  émet ensuite l'information  $\sigma(a_j)$  à tous les neurones de la couche suivante  $k + 1$  auxquels il est connecté, où  $\sigma$  est une fonction d'activation dont on verra les propriétés dans la section 9.2.1.

Enfin, la dernière couche émet le vecteur ou le scalaire de sortie du réseau. Sa nature exacte dépend du problème traité (classification ou régression); cela sera abordé à la section 9.2.2.

L'apprentissage consiste à adapter les paramètres (poids et biais) du réseau de manière à prédire correctement les étiquettes d'une base d'exemples. Il est l'objet de la section 9.4. Avant de traiter l'apprentissage, on aura évoqué en section 9.3 quelques résultats théoriques permettant de se représenter ce que peuvent calculer les réseaux de neurones.

*Remarque.* Dans la suite de ce chapitre, nous utilisons les notations du livre *Pattern recognition and machine learning* de C. Bishop :  $z$  est le signal en sortie d'un neurone appartenant à une couche cachée (donc *après* activation), et  $a$  est la somme pondérée des signaux en entrée d'un neurone (donc *avant* activation). Notez que la convention inverse est adoptée sur la page web suivante : <https://en.wikipedia.org/wiki/Backpropagation>

### 9.2.1 Fonction d'activation $\sigma$ d'un neurone

Le signal en sortie d'un neurone  $j$  est :

$$z_j = \sigma(a_j) = \sigma\left(\sum_i w_{ji} z_i + b_j\right)$$

où la somme porte sur les neurones de la couche précédente connectés au neurone considéré, chacun ayant une sortie  $z_i$ , les poids des connexions étant  $w_{ji}$  et le biais  $b_j$ . Il est fondamental de considérer des activations non linéaires. Un réseau de neurones avec des activations linéaires ne ferait que des compositions de fonctions linéaires des entrées. Un tel réseau émettrait donc en sortie une combinaison linéaire des entrées, ce qui limiterait grandement son intérêt.

Une activation en échelon comme pour le perceptron de Rosenblatt restreint les signaux  $z_i$  aux seules valeurs 0 ou 1. Différentes activations ont été proposées. Toutes sont continues et croissantes, et sont (presque partout) dérivables pour des raisons qui apparaîtront dans la suite de ce chapitre. La figure 9.6 représente certaines des fonctions d'activation définies ci-dessous.

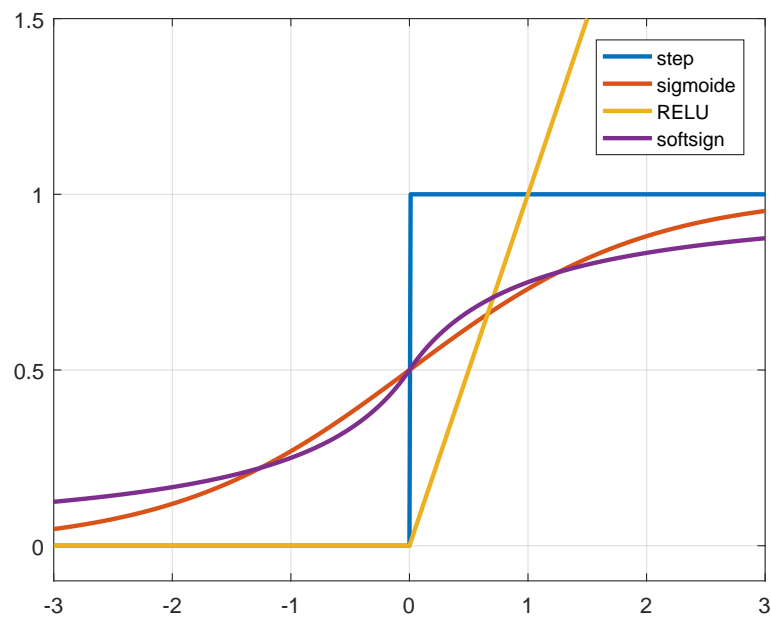


FIGURE 9.6 – Quelques fonctions d'activation.

|                           |  |
|---------------------------|--|
| step<br>(voir perceptron) | $s(a) = \begin{cases} 0 & \text{si } a < 0 \\ 1 & \text{si } a \geq 0 \end{cases}$ |
| sigmoïde                  | $\sigma(a) = \frac{1}{1+e^{-a}}$   |
| tangente hyperbolique     | $\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} = 2\sigma(2a) - 1$                   |
| Rectified Linear Unit     | $\text{ReLU}(a) = \max\{0, a\}$  |
| softsign                  | $\text{softsign}(a) = \frac{a}{1+ a }$   |

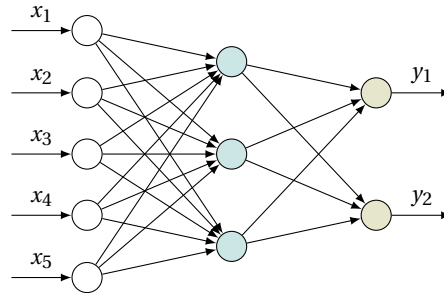
## 9.2.2 Sortie d'un réseau de neurones

Étant donnée une observation  $\mathbf{x} \in \mathbb{R}^d$ , la sortie d'un réseau de neurones possédant  $d$  neurones en entrée se calcule en propageant les calculs « de gauche à droite » dans la figure 9.5. La forme de la sortie change selon le problème d'apprentissage que l'on souhaite traiter.

### 9.2.2.1 Problèmes de régression

Dans les problèmes de régression, les étiquettes des observations sont des scalaires ou des vecteurs. Un réseau de neurones adapté à un problème de régression a autant de neu-

rones de sortie que la dimension des étiquettes à prédire, comme représenté ci-dessous. Dans cet exemple, le problème de régression consiste à prédire  $\mathbf{y} \in \mathbb{R}^2$  à partir de  $\mathbf{x} \in \mathbb{R}^5$  avec une seule couche cachée de trois neurones. Pour un problème de régression dans lequel on cherche à prédire une valeur réelle, il n'y a qu'un seul neurone de sortie. Bien entendu, l'architecture est à adapter au problème traité.



Il n'y a pas de fonction d'activation sur les neurones de la couche de sortie : avec les notations habituelles, la  $j$ -ème composante de sortie est :

$$y_j = \sum_i w_{ji} z_i + b_j$$

où  $z_i$  est le signal émis par le neurone  $i$  de la couche cachée.

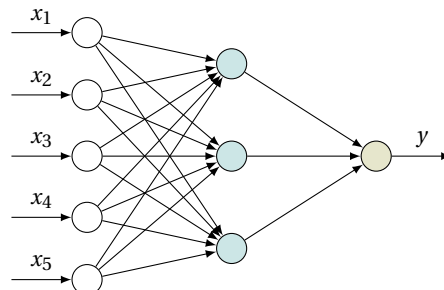
Dans notre exemple, la sortie  $y_j$  s'expriment en fonction des entrées du réseau par la relation :

$$y_j = \sum_{i=1}^3 w_{ji} \sigma \left( \sum_{l=1}^5 w_{il}^* x_l + b_i^* \right) + b_j$$

où  $\sigma$  est l'activation des neurones cachés, et  $w^*$  et  $b^*$  désignent leurs poids et biais.

### 9.2.2.2 Problèmes de classification bichasse

Dans les problèmes de classification où il faut distinguer deux classes, les étiquettes sont 1 ou 0; elles correspondent respectivement aux classes  $\mathcal{C}_1$  ou  $\mathcal{C}_2$ . Un exemple de réseau de neurones pour la classification bichasse d'observations  $\mathbf{x} \in \mathbb{R}^5$  est représenté ci-dessous.



Ce réseau a un seul neurone de sortie, de fonction d'activation  $\sigma$ , la valeur émise étant :

$$y = \sigma \left( \sum_i w_i z_i + b \right)$$

en fonction des poids  $w_i$  reliant le neurone de sortie à ceux de la couche cachée et des signaux  $z_i$  émis par les neurones de la dernière couche cachée.

Pour obtenir une probabilité a posteriori  $p(\mathcal{C}_1 | \mathbf{x})$  (et pour permettre l'apprentissage, comme on le verra plus loin), on utilise une fonction d'activation  $\sigma$  sur le neurone de sortie de manière à normaliser la sortie à une valeur entre 0 et 1, supérieure à 1/2 pour la classe  $\mathcal{C}_1$  et inférieure à 1/2 pour la classe  $\mathcal{C}_2$ . Il est courant de choisir la fonction sigmoïde. La fonction d'activation des neurones des couches cachées n'est pas nécessairement comprise entre 0 et 1, et peut donc être choisie différemment.

La règle de décision est alors :

- si  $y > 1/2$ , classification dans classe  $\mathcal{C}_1$  ;
- si  $y < 1/2$ , dans  $\mathcal{C}_2$ .

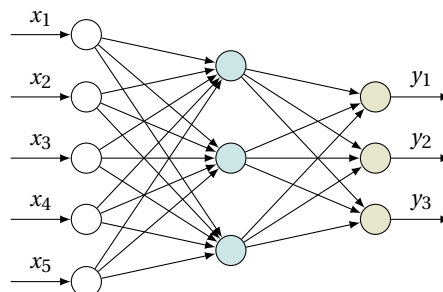
ou de manière équivalente (notons la similitude avec la régression logistique lorsque  $\sigma$  est la fonction sigmoïde) :

- si  $\sum_i w_i z_i + b > 0$ , classification dans la classe  $\mathcal{C}_1$  ;
- si  $\sum_i w_i z_i + b < 0$ , dans  $\mathcal{C}_2$ .

### 9.2.2.3 Problèmes de classification multiclasse

On suppose que  $K > 2$  classes sont à discriminer. Il faut encoder les classes par des étiquettes scalaires, mais choisir arbitrairement des numéros de classes ( $y = 2$ ,  $y = 5$ , etc.) n'a pas vraiment de sens. On utilise un encodage particulier des étiquettes : l'encodage « un parmi  $n$  », ou *one-hot encoding*. Cela consiste à définir l'étiquette  $y$  comme un vecteur de dimension  $K$ , tel que la composante  $y_i$  est nulle sauf en l'indice correspondant à la classe à coder pour laquelle  $y_i = 1$ .

L'exemple ci-dessous montre un réseau de neurones adapté à un problème de classification en  $K = 3$  catégories à partir d'observations dans  $\mathbb{R}^5$ .





Dans cet exemple, la classe 2 est encodé par l'étiquette (0, 1, 0).

La règle de décision est alors d'associer l'observation à la classe  $i$  telle que  $y_i$  a la plus grande valeur parmi les sorties.

Comme dans le cas bicolore, on fait apparaître des probabilités a posteriori en normalisant les sorties entre 0 et 1. On utilise la règle dite « SoftMax » sur chaque sortie  $y_i$  :

$$y_i = \frac{1}{\sum_{k=1}^K \exp(a_k)} \exp(a_i)$$

où chaque  $a_k$  représente le signal en entrée du neurone de sortie  $k$ , qui est une combinaison linéaire des signaux émis par les neurones de la couche précédente.

Cette opération a pour effet d'« écarter » numériquement la plus grande valeur des autres, et de normaliser les  $y_i$  entre 0 et 1 (d'où l'expression SoftMax : il s'agit d'une version « douce » du maximum qui consisterait à mettre toutes les valeurs à 0 sauf la valeur maximale à 1). Par ailleurs, on est assuré que  $\sum_i y_i = 1$ .

Il ne s'agit pas d'une fonction d'activation à proprement parler car la normalisation est commune à tous les neurones de sortie. On parle de « couche SoftMax » (*SoftMax layer*) en sortie.

### 9.3 Expressivité des réseaux de neurones artificiels

Avant d'aborder l'apprentissage, nous discutons de l'« expressivité » des réseaux de neurones artificiels : quelles fonctions peuvent bien être calculées par ces modèles ?

#### 9.3.1 Théorème d'approximation universelle

Commençons par discuter le réseau de neurones artificiels le plus simple, possédant une couche cachée. Supposons  $d$  neurones en entrée lisant  $\mathbf{x} \in \mathbb{R}^d$ ,  $M$  neurones dans la couche cachée (fonction d'activation  $\sigma$ ), et un neurone en sortie émettant  $F(\mathbf{x}) \in \mathbb{R}$ . Il s'agit d'une architecture pour un problème de régression dans  $\mathbb{R}$ .

Les règles de propagation permettent d'écrire l'expression de la sortie  $F(\mathbf{x})$  en fonction de l'entrée  $\mathbf{x}$  :

$$F(\mathbf{x}) = \sum_{k=1}^M w_k^s \sigma(\mathbf{w}_k \cdot \mathbf{x} + b_k) + b$$

où  $\forall k \in \{1, \dots, M\}$ ,  $\mathbf{w}_k \in \mathbb{R}^d$  représente le vecteur des poids reliant les  $d$  entrées au  $k$ -ème neurone de la couche cachée,  $b_k \in \mathbb{R}$  le biais correspondant, et  $\forall k \in \{1, \dots, M\}$ ,  $w_k^s \in \mathbb{R}$  est le poids entre ce  $k$ -ème neurone caché et le neurone de sortie, et  $b$  le biais du neurone de sortie.

Le théorème suivant nous indique ce qu'il est possible de calculer avec une fonction de cette forme.

**Théorème 9.3** *Théorème de Cybenko (1989) (et autres chercheurs), ou théorème d'approximation universelle.*

Soient  $\phi$  une fonction strictement croissante continue bornée, et  $C$  un compact de  $\mathbb{R}^d$ .

Pour tout  $\varepsilon > 0$  et  $f$  continue sur  $C$ , il existe  $M \in \mathbb{N}$ , et pour tout  $i \in \{1, \dots, M\}$ ,  $v_i, b_i \in \mathbb{R}$ ,  $\mathbf{w}_i \in \mathbb{R}^d$ , tels que la fonction :

$$F(\mathbf{x}) = \sum_{i=1}^M v_i \phi(\mathbf{w}_i \cdot \mathbf{x} + b_i)$$

vérifie :

$$\forall \mathbf{x} \in C, |F(\mathbf{x}) - f(\mathbf{x})| < \varepsilon$$

Des théorèmes équivalents ont été démontrés pour certaines fonctions  $\phi$  non bornées comme la fonction ReLU. Le théorème de Cybenko nous dit que toute fonction réelle continue sur un compact peut être approchée d'aussi près que l'on veut par la sortie d'un perceptron à une couche cachée possédant  $M$  neurones. La fonction  $\phi$  joue le rôle de l'activation  $\sigma$  qui en possède bien les propriétés (qu'elle soit bornée comme la sigmoïde, ou non-bornée en ajoutant certaines hypothèses).

On se rend compte que la non-linéarité de l'activation joue un rôle essentiel : si  $\phi$  était une fonction affine, alors  $F(\mathbf{x})$  serait également affine. Bien entendu les fonctions affines ne peuvent pas approcher toute fonction continue sur un compact à une précision arbitraire.

*Remarque.* Il s'agit d'un résultat d'existence (pour les matheux, il s'agit d'une variante du théorème de Stone-Weierstrass). Le théorème ne dit pas comment construire le réseau (choix de  $M$ , qui est potentiellement grand) ni comment fixer les poids pour approcher une fonction donnée avec une précision  $\varepsilon$  donnée. De fait, les techniques modernes de *deep learning* privilégient un grand nombre de couches cachées, architecture plus adaptée à l'apprentissage qu'une seule couche cachée faite d'un très grand nombre de neurones.

### 9.3.2 Réseaux de neurones et classifieur universel

Le théorème précédent nous permet de dire que les réseaux de neurones sont également des classifieurs universels, dans le sens suivant.

Supposons que l'espace des observations  $C$  (supposé compact) soit partitionné en l'union des  $K$  classes à identifier  $C = \mathcal{C}_1 \cup \dots \cup \mathcal{C}_K$ .

Chaque observation  $\mathbf{x}$  appartenant à une des classes, on introduit la fonction de classification multiclasse :  $f(\mathbf{x}) = k$  si  $\mathbf{x} \in \mathcal{C}_k$ . On ne peut pas appliquer le théorème de Cybenko à  $f$  car cette fonction n'est pas continue.

Néanmoins, le théorème de Lusin nous indique que, pour  $\varepsilon > 0$ , il existe un ensemble  $D \subset C$  de mesure  $\lambda(D) \geq (1 - \varepsilon)\lambda(C)$  tel que  $f$  coïncide sur  $D$  avec une fonction  $g$  continue.

On peut appliquer le théorème de Cybenko à  $g$ , ce qui nous permet de conclure : il existe un réseau de neurones qui approche  $f$  à  $\varepsilon$  près, sauf sur un ensemble de mesure au plus  $\varepsilon\lambda(C)$ .

Ce réseau de neurones fournit des classifications incorrectes, mais la mesure de l'ensemble des observations incorrectement classées est contrôlée, et il existe un réseau pour lequel cette mesure peut être rendue aussi petite que l'on veut.

*Remarque.* Certes, les réseaux de neurones pour la classification n'ont pas cette architecture (cf. la couche de sortie qui emploie une sigmoïde ou est une couche SoftMax). Néanmoins, il ne faut pas perdre de vue qu'il ne s'agit de toute façon que d'un résultat d'existence. L'enseignement principal est que les réseaux de neurones ont une expressivité suffisante pour représenter correctement toutes les frontières de classification. Ce n'est pas le cas par exemple du perceptron de Rosenblatt ou du classifieur de la régression logistique qui sont restreints aux frontières linéaires.

## 9.4 Apprentissage et rétropropagation

L'apprentissage des poids d'un réseau de neurones en fonction d'une base d'exemples se fait en minimisant un coût d'erreur global par descente de gradient (voir la section B.3 en annexe). On parle aussi d'entraînement du réseau (*training*). Le calcul exact du gradient, c'est-à-dire sans utiliser de schéma numérique approché, est possible grâce à un algorithme astucieux popularisé dans les années 1980 : la « rétropropagation des erreurs » (*backpropagation*).

Pour alléger les notations, on suppose que la première composante de chaque observation est égale à 1, ce qui permet de traiter les biais comme des poids pour simplifier les équations.

### 9.4.1 Notations

$(\mathbf{x}_n, y_n)_{n=1\dots N}$  désigne la base d'apprentissage faite de  $N$  observations étiquetées. Les  $\mathbf{x}_n$  sont des vecteurs de dimensions  $d$  ( $d$  neurones en entrée), et les étiquettes  $y_n$  sont, selon les situations :

- pour un problème de régression :  $y_n = (y_{n1}, \dots, y_{ns}) \in \mathbb{R}^s$  ( $s \geq 1$  neurones de sortie), cf section 9.2.2.1 ;
- pour un problème de classification biclasse :  $y_n = 0$  ou  $1$  (un neurone en sortie avec une activation  $\sigma$ ), cf section 9.2.2.2 ;
- pour un problème de classification à  $K > 2$  classes  $y_n$  est un vecteur de dimension  $K$  *one-hot encoding* fait de 0 et un seul 1 ( $s = K$  neurones de sorties dans une couche SoftMax), cf section 9.2.2.3.

La sortie du perceptron sur l'entrée  $\mathbf{x}_n = (x_{n1}, \dots, x_{nd})$  est notée :  $\hat{y}_n = (\hat{y}_{n1}, \dots, \hat{y}_{ns})$ . Dans tous les cas, le but ultime est que la prédiction  $\hat{y}_n$  ne soit pas très éloigné de l'étiquette d'apprentissage  $y_n$ .

### 9.4.2 Coût d'erreur

Le risque empirique (coût d'erreur moyen sur la base d'apprentissage) est la moyenne des coûts des erreurs  $E_n$  sur chaque observation étiquetée  $(\mathbf{x}_n, y_n)$  :

$$E(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N E_n(\mathbf{w})$$

Il s'agit d'une fonction de  $\mathbf{w}$ , l'ensemble des poids du réseau.

Il faut choisir la fonction  $E_n$  en fonction du problème traité. Les choix les plus courants sont :

— pour un problème de régression :

$$E_n(\mathbf{w}) = \frac{1}{2} \|y_n - \hat{y}_n\|_2^2$$

— pour un problème de classification biclasse :

$$E_n(\mathbf{w}) = -y_n \log(\hat{y}_n) - (1 - y_n) \log(1 - \hat{y}_n)$$

— pour un problème de classification à  $K > 2$  classes :

$$E_n(\mathbf{w}) = - \sum_{j=1}^K y_{nj} \log(\hat{y}_{nj})$$

Ce coût est l'entropie croisée entre les distributions de probabilité discrètes  $(y_{n1}, \dots, y_{nK})$  et  $(\hat{y}_{n1}, \dots, \hat{y}_{nK})$ . Par convention :  $0 \log(0) = 0$ .

Dans tous les cas,  $E_n$  est positif (car si  $x \leq 1$ ,  $-\log(x) \geq 0$ ) et est minimal (et nul) lorsque  $y_n = \hat{y}_n$ . En effet, dans le cas de la classification biclasse,  $y_n = 0$  ou  $y_n = 1$  donc  $E_n$  est minimal lorsque  $\hat{y}_n = y_n$ , et dans le cas de  $K > 2$  classes, la convention sur le logarithme et l'utilisation du *one-hot encoding* font que seul un terme est non-nul dans la somme.

L'objectif de l'apprentissage est de trouver les poids  $\mathbf{w}$  qui minimisent l'erreur moyenne  $E(\mathbf{w})$ .

*Remarque.* Dans le cas de la classification, la fonction mesurant le coût d'erreur (*loss*) n'étant pas un coût 0-1, l'erreur  $E$  n'est pas directement liée à la proportion de fausses classifications.

*Remarque.* On note que le classifieur biclasse de la régression logistique est équivalent à un réseau de neurones sans couche cachée (à faire en exercice). Les paramètres ne sont pas estimés numériquement de la même manière dans les deux cas, mais l'unicité de l'optimum fait que l'on doit trouver la même solution, aux erreurs numériques près.

### 9.4.3 Descente de gradient et perceptron multicouche

L'annexe B.3 nous rappelle que l'algorithme de descente de gradient permettant de minimiser le risque empirique  $E(\mathbf{w})$  met à jour chaque poids  $w_{ji}$  (poids du neurone  $i$  de la couche  $k$  vers le neurone  $j$  de la couche  $k + 1$ ) de la manière suivante :

$$w_{ji} \leftarrow w_{ji} - \eta \frac{\partial E}{\partial w_{ji}}(\mathbf{w})$$

où le paramètre positif  $\eta$  est le pas de descente, appelé « taux d'apprentissage » lorsqu'on entraîne un classifieur ou régresseur.

Par linéarité de la dérivation :

$$\frac{\partial E}{\partial w_{ji}} = \frac{1}{N} \sum_{n=1}^N \frac{\partial E_n}{\partial w_{ji}}(\mathbf{w})$$

Nous allons donc commencer par calculer les dérivées partielles  $\partial E_n / \partial w_{ji}$ .

Le signal arrivant au neurone  $j$  est :

$$a_j = \sum_i w_{ji} z_i$$

où  $z_i (= \sigma(a_i))$  est le signal sorti par le neurone  $i$  connecté à  $j$

La formule de dérivation composée (voir l'annexe A.2) nous indique (ici, avec les notations de cette annexe,  $a_j = g(w_{ji}) \in \mathbb{R}$ ) :

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}$$

On a  $\frac{\partial a_j}{\partial w_{ji}} = z_i$ , donc, en posant  $\delta_j = \frac{\partial E_n}{\partial a_j}$  :

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

Il s'avère qu'il existe une astuce pour calculer efficacement les  $\delta_j$  en chaque neurone : l'algorithme de rétropropagation des erreurs (*backpropagation*). Il s'appuie sur le calcul des  $\delta_j$  aux neurones de sortie, puis calcule, de proche en proche, les  $\delta_j$  des couches cachées jusqu'aux neurones de la couche cachée la plus proche de l'entrée du réseau.

#### 9.4.3.1 Calcul de $\delta_j$ aux neurones de sortie

Nous montrons qu'aux neurones de la couche de sortie,  $\delta_j = \frac{\partial E_n}{\partial a_j}$  est calculable en fonction des sorties  $\hat{y}$  du réseau, qui dépendent bien entendu de la fonction de coût adoptée.

Pour reprendre les fonctions de coût introduite précédemment :

- problème de régression :  $E_n = \frac{1}{2} \|y_n - \hat{y}_n\|_2^2$  où  $\hat{y}_n = (a_l)_{1 \leq l \leq s}$  est le vecteur des sorties de chacun des  $s$  neurones de la dernière couche.

Par conséquent, aux neurones de sortie :

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \hat{y}_{nj} - y_{nj}$$

- problème de classification biclasse :  $E_n(\mathbf{w}) = -y_n \log(\hat{y}_n) - (1 - y_n) \log(1 - \hat{y}_n)$  où  $\hat{y}_n = \sigma(a)$  est la sortie du neurone de la dernière couche, avec  $\sigma$  la fonction sigmoïde.

Comme  $\sigma$  vérifie  $\log(\sigma(a)) = -\log(1 + e^{-a})$  et  $\log(1 - \sigma(a)) = -a - \log(1 + e^{-a})$ , on peut écrire  $E_n = y_n \log(1 + e^{-a}) + (1 - y_n)a + (1 - y_n) \log(1 + e^{-a}) = a - y_n a + \log(1 + e^{-a})$ , donc :

$$\delta = \frac{\partial E_n}{\partial a} = 1 - y_n - \frac{e^{-a}}{1 + e^{-a}} = \frac{1}{1 + e^{-a}} - y_n$$

soit :

$$\delta = \hat{y}_n - y_n$$

- problème de classification à  $s > 2$  classes :  $E_n = -\sum_{l=1}^s y_{nl} \log(\hat{y}_{nl})$  où  $\hat{y}_{nl} = e^{a_l} / \sum_m e^{a_m}$  (couche SoftMax). Autrement écrit :

$$E_n = -\sum_{l=1}^s y_{nl} \left( a_l - \log\left(\sum_m e^{a_m}\right) \right)$$

Par conséquent :

$$\delta_j = \frac{\partial E_n}{\partial a_j} = -\left( y_{nj} - \frac{e^{a_j}}{\sum_m e^{a_m}} \sum_{l=1}^s y_{nl} \right)$$

Comme pour tout  $n$ ,  $\sum_l y_{nl} = 1$  (cf *one-hot encoding*), on peut conclure. Au neurone de sortie  $j$  :

$$\delta_j = \hat{y}_{nj} - y_{nj}$$

Pour toutes les fonctions de coût vues ici, on voit que  $\delta_j$  est calculé en un neurone de sortie comme la différence entre la sortie attendue  $y$  et la sortie observée  $\hat{y}$ .

### 9.4.3.2 Calcul de $\delta_j$ aux neurones cachés

On calcule  $\delta_j$  en un neurone  $j$  appartenant à une couche  $k$ . On applique la règle de dérivation composée (annexe A.2) en exprimant la dépendance de  $E_n$  par rapport au vecteur des  $(a_l)$  représentant les entrées des neurones de la couche  $k + 1$  liés au neurone  $j$  :

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_l \frac{\partial E_n}{\partial a_l} \frac{\partial a_l}{\partial a_j}$$

Ici,  $l$  parcourt l'ensemble des neurones de la couche  $k + 1$  auxquels est connecté  $j$  de la couche  $k$ .

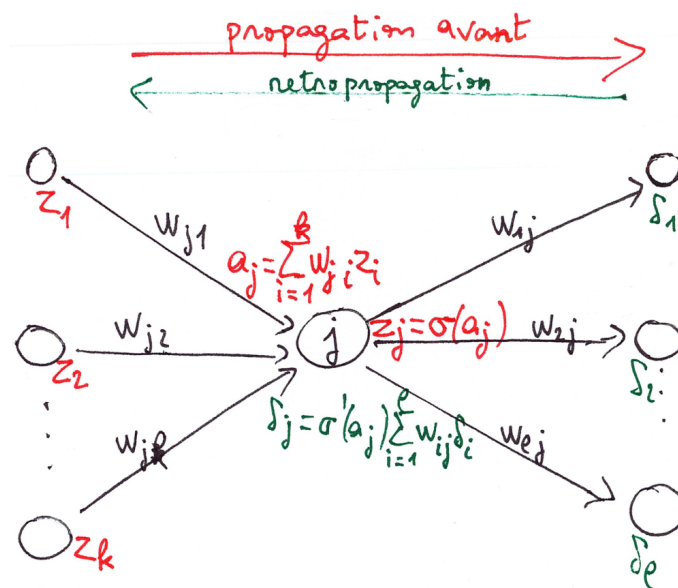


FIGURE 9.7 – Illustration des deux étapes successives de propagation avant puis rétropropagation pour calculer les  $z$  et  $\delta$  intervenant dans la dérivée partielle  $\partial E / \partial w_{ji}$ . Dans cette figure, on a isolé le neurone  $j$ , et représenté uniquement les neurones qui lui sont connectés dans la couche précédente et dans la couche suivante.

De plus :

$$\frac{\partial a_l}{\partial a_j} = \frac{\partial}{\partial a_j} \sum_i w_{li} \sigma(a_i) = w_{lj} \sigma'(a_j)$$

où  $i$  parcourt l'ensemble des neurones de la couche  $k$  auquel le neurone  $l$  de la couche  $k+1$  est connecté : le neurone  $j$  est l'un des  $i$ , les autres ne dépendent pas des  $a_j$ .  $\sigma'$  désigne la dérivée de l'activation  $\sigma$ .

On a donc obtenu la formule :

$$\delta_j = \sigma'(a_j) \sum_l w_{lj} \delta_l$$

### 9.4.3.3 Conclusion : calcul des $\delta_j$

On peut calculer tous les  $\delta$  sur le réseau par *rétropropagation des erreurs*. On commence par calculer les erreurs comme différence entre sortie attendue  $y_n$  et sortie du réseau  $\hat{y}_n$  (paragraphe 9.4.3.1), puis on propage les  $\delta_j$  vers les couches d'entrée à l'aide de la formule concluant le paragraphe 9.4.3.2.

L'algorithme de calcul des  $\delta_j$  avec propagation avant suivie de rétropropagation est illustré par la figure 9.7.

### 9.4.4 Apprentissage et rétropropagation des erreurs

L'apprentissage (c'est-à-dire l'adaptation des poids  $\mathbf{w}$  du réseau de manière à minimiser le coût moyen d'erreur  $E(\mathbf{w})$ ) peut donc se faire à l'aide d'un algorithme de descente de gradient, en itérant les deux étapes suivantes jusqu'à satisfaction d'un critère d'arrêt :

1. pour chaque observation étiquetée  $(\mathbf{x}_n, y_n)$  en entrée :
  - (a) propagation avant pour calculer les  $a_j$  et  $z_j$  en chaque neurone, ainsi que la sortie  $\hat{y}_n$  du réseau;
  - (b) calcul des  $\delta_j$  aux neurones de sortie :  $\delta_j = \hat{y}_{nj} - y_{nj}$ ;
  - (c) rétropropagation des  $\delta$  : en chaque neurone caché, calcul de  $\delta_j = \sigma'(a_j) \sum_l w_{lj} \delta_l$
  - (d) calcul des dérivées partielles :  $\frac{\partial E_n}{\partial w_{ji}}(\mathbf{w}) = \delta_j z_i$
2. pour chaque poids  $w_{ji}$ , calcul de  $\frac{\partial E}{\partial w_{ji}}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N \frac{\partial E_n}{\partial w_{ji}}(\mathbf{w})$  et mise à jour :

$$w_{ji} \leftarrow w_{ji} - \eta \frac{\partial E}{\partial w_{ji}}(\mathbf{w})$$

En pratique, il est coûteux en temps de calcul et occupation mémoire de ne faire la mise à jour dans l'étape 2 qu'une fois que l'intégralité de la base d'exemples a été parcourue dans l'étape 1.

Deux variantes sont utilisées dans les implantations logicielles :

- version *online* / *algorithme du gradient stochastique* : mise à jour des poids pour *chaque* observation de la base d'apprentissage, selon la règle :

$$w_{ji} \leftarrow w_{ji} - \eta \frac{\partial E_n}{\partial w_{ji}}$$

- version *mini-batch* / *par mini-lot* : mise à jour des poids après parcours d'un sous-ensemble  $B_m$  d'observations :

$$w_{ji} \leftarrow w_{ji} - \eta \sum_{n \in B_m} \frac{\partial E_n}{\partial w_{ji}}$$

tel que  $\cup_m B_m = \{1, \dots, N\}$

Pour faciliter la lecture, la discussion détaillée de ces algorithmes est renvoyée en annexe (section B.3.3), notamment les méthodes de moment utilisées dans les bibliothèques logicielles pour accélérer la convergence.

### 9.4.5 Remarques

La rétropropagation est une astuce pour calculer le gradient  $\nabla E$ , ce n'est pas l'apprentissage proprement dit, même si par abus de langage on peut parler d'« apprentissage par rétropropagation ».



L'algorithme d'apprentissage présenté ici est l'implantation d'une descente de gradient. Pour résoudre numériquement un problème d'optimisation, la descente de gradient à pas fixe est une méthode simple mais assez grossière. Plusieurs méthodes existent pour améliorer l'optimisation du risque empirique en évitant d'échouer dans des minima locaux ou en améliorant la vitesse de convergence. Ces méthodes sont implantées dans les bibliothèques logicielles. Dans tous les cas, le gradient reste utile et est calculé par rétropropagation.

On voit que toutes les formules employées sont exactes, et nécessitent des fonctions d'activation dérivables. Au passage, on remarque que les neurones peuvent avoir chacun leur propre fonction d'activation.

La formule de rétropropagation  $\delta_j = \sigma'(a_j) \sum_l w_{lj} \delta_l$  montre qu'en remontant de couche en couche, on multiplie par la dérivée de la fonction d'activation. Or, pour la sigmoïde, la dérivée s'exprime comme  $\sigma'(t) = \sigma(t)(1 - \sigma(t))$ , qui est toujours inférieur à 1/4. Le gradient est donc de plus en plus petit au fur et à mesure que l'on propage les calculs de rétropropagation vers les couches d'entrée. Les poids des neurones près de l'entrée du réseau évoluent donc peu dans l'apprentissage et nécessitent un long entraînement. On parle d'« évanouissement du gradient » (*vanishing gradient*), et le phénomène est d'autant plus marqué lorsque le réseau a de nombreuses couches. Cela motive l'utilisation de l'activation ReLU (*rectified linear unit*) en apprentissage profond (cf. chapitre 10). Cette fonction d'activation est dérivable (sauf en l'unique point 0), et sa dérivée (0 pour  $x < 0$  et 1 pour  $x > 0$ ) ne s'« évanouit » pas par multiplications successives.

## 9.5 Problème du sur-apprentissage

L'apprentissage (entraînement) d'un réseau de neurones consiste à minimiser l'erreur empirique. On est donc susceptible de se retrouver dans une situation de sur-apprentissage, comme illustré par la figure 9.8.

On peut envisager deux manières d'éviter le sur-apprentissage :

- arrêt prématuré de l'entraînement (*early stopping*). On arrête l'apprentissage quand l'erreur sur une base d'observations de test (ou base de validation), indépendante de la base d'apprentissage, commence à augmenter. On peut aussi tout simplement arrêter l'apprentissage après un nombre d'époques fixé a priori;
- régularisation. On contrôle la « complexité » du réseau (en fait l'intervalle de variation des poids) en cherchant plutôt à minimiser :

$$E(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2$$

avec un hyperparamètre  $\lambda > 0$ . On remarque que cela ajoute le terme  $2\lambda w_{ji}$  dans la dérivée partielle  $\partial E / \partial w_{ji}$  de l'algorithme d'apprentissage, ce qui ne pose pas de difficulté particulière.

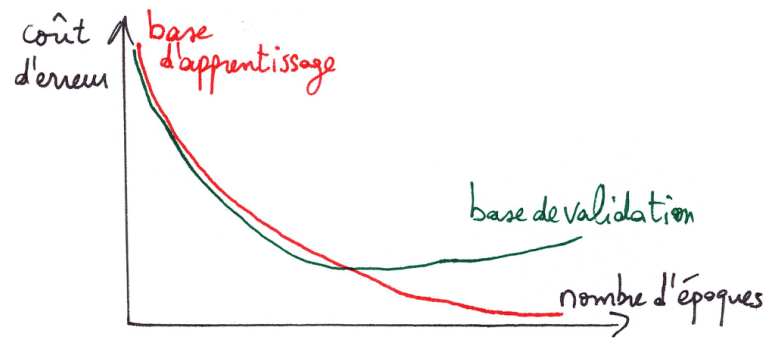


FIGURE 9.8 – Illustration du sur-apprentissage. Coût d'erreur en fonction du nombre d'époques d'apprentissage, calculé sur la base d'entraînement et sur une base de test (validation) indépendante. Le sur-apprentissage se traduit par un coût d'erreur sur la base de validation qui finit par croître, alors qu'il continue logiquement à diminuer sur la base d'apprentissage.

## 9.6 Conclusion sur les réseaux de neurones pré-2012...

Les réseaux de neurones artificiels ou perceptrons multicouches permettent d'aller au-delà des classifieurs linéaires (perceptron de Rosenblatt, régression logistique, SVM linéaire, etc.) en introduisant des non-linéarités par l'intermédiaire des fonctions d'activation. On a vu qu'ils sont des approximateurs et classifieurs universels. Si ce résultat n'a essentiellement qu'un intérêt théorique, il permet aussi de se rendre compte qu'approcher finement une fonction peut être au prix d'une grande augmentation de la complexité de l'architecture (nombre de neurones cachés).

L'algorithme de rétropropagation des erreurs permet la mise en œuvre de l'apprentissage des poids du réseau par minimisation de l'erreur empirique à l'aide d'une descente de gradient.

Il ne faut pas perdre de vue les nombreux paramètres restant à fixer par l'utilisateur :

- l'architecture du réseau : combien de couches cachées? combien de neurones dans chaque couche?
- le nombre d'époques dans l'apprentissage en cas d'arrêt prématuré, ou le paramètre de régularisation  $\lambda$ ;
- le taux d'apprentissage  $\eta$  qui peut être amené à varier (intuitivement, on peut commencer l'apprentissage avec une grande valeur de  $\eta$  puis diminuer cette valeur lorsque la descente de gradient commence à converger);
- les valeurs initiales des poids et biais (l'initialisation est assez critique pour les performances d'apprentissage : elle est généralement aléatoire et différentes stratégies existent);
- le critère d'arrêt de la descente de gradient;
- les tailles de lots (*batches*) dans l'algorithme du gradient stochastique par lot.

S'il peut être possible de s'aider de la validation croisée, au prix d'un temps de calcul parfois rédhibitoire, une large partie des réglages reste tributaire de l'expérience de l'utilisateur.

## 9.7 Pour approfondir...

**Théorème d'approximation universelle** Une démonstration du théorème de Cybenko se trouve dans son article :

G. Cybenko, *Approximation by Superpositions of a Sigmoidal Function*, Mathematics of Control, Signals and Systems, Vol. 2, No. 4, p. 303-314, 1989.

Des articles scientifiques récents démontrent des théorèmes d'approximation universelle en supposant que le nombre de neurones dans chaque couche cachée est borné, mais en permettant un certain nombre de couches cachées. D'une certaine manière, ces résultats expliquent le succès de l'apprentissage profond qui fera l'objet du chapitre suivant.

Voir par exemple les articles cités ici :

[https://en.wikipedia.org/wiki/Universal\\_approximation\\_theorem](https://en.wikipedia.org/wiki/Universal_approximation_theorem)



# Introduction aux réseaux de neurones convolutifs et à l'apprentissage profond

Pour conclure ce cours, nous souhaitons donner quelques éléments permettant de comprendre les enjeux de l'apprentissage profond (*deep learning*).

## 10.1 Le retour des réseaux de neurones

En 2012, les participants au défi de reconnaissance d'images ImageNet ont la surprise de voir une équipe de l'université de Toronto écraser la concurrence avec des taux d'erreurs de classification passant subitement de 26% à 15% : les résultats sont disponibles en ligne à cette URL :

<http://www.image-net.org/challenges/LSVRC/2012/results.html>

La description du classifieur vainqueur, le réseau AlexNet (par Alex Krizhevsky, Ilya Sutskever, et Geoffrey Hinton de l'Université de Toronto), est la suivante :

“Our model is a large, deep convolutional neural network trained on raw RGB pixel values. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three globally-connected layers with a final 1000-way softmax. It was trained on two NVIDIA GPUs for about a week. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of convolutional nets. To reduce overfitting in the globally-connected layers we employed hidden-unit "dropout", a recently-developed regularization method that proved to be very effective.”

C'est le début de la révolution de l'apprentissage profond. Par comparaison aux réseaux de neurones classiques qui étaient quelque peu tombés en désuétude, les nouveaux réseaux superposent plusieurs couches cachées. Ils dépendent donc d'un grand nombre de paramètres. Ils restent entraînaibles en pratique, pour plusieurs raisons :

- les poids de certaines couches cachées ne sont pas quelconques, mais correspondent à des convolutions : on parle de réseaux de neurones convolutifs (*convolutional neural*

- networks*, CNN). Cela permet de réduire le nombre de paramètres à estimer ;
- l’entraînement tire profit du grand nombre de données disponibles à l’ère d’Internet : le défi ImageNet 2012 propose une base d’entraînement de 1,2 millions d’images dans 1000 catégories, et une base de test de 150 000 images. Plus de 14 millions d’images sont incluses dans ImageNet à l’heure actuelle (voir : <http://www.image-net.org/>) ;
  - les chercheurs se sont rendu compte qu’ils disposaient dans leur machine de processeurs bon marché adaptés au calcul matriciel efficace ou aux opérations de convolution qui sont la base du traitement des images : les GPU (*graphics processing unit*) embarqués sur les cartes graphiques utilisées pour les jeux vidéo (et donc de coût modeste).

Si les réseaux convolutifs ont été introduits par Yann Le Cun et ses co-auteurs dans les années 1990, c’est bien la disponibilité de grandes bases d’apprentissage, l’amélioration du matériel, ainsi que le développement de bibliothèques logicielles qui ont permis la révolution de l’apprentissage profond. Notons que la notion de « profondeur » est relative : dans certaines applications on se satisfait d’un réseau possédant seulement quelques couches de convolution.

## 10.2 Réseaux de neurones convolutifs

Nous allons essayer de donner une idée des différents ingrédients qui entrent dans la composition des modèles de l’apprentissage profond tels qu’AlexNet décrit plus haut. La figure 10.1 représente le réseau AlexNet de 2012, ainsi que le réseau LeNet (développé de 1989 à 1995 par Yann LeCun et ses co-auteurs).

### 10.2.1 Convolution

Le traitement du signal est la discipline scientifique qui s’intéresse aux signaux numériques ou analogiques et à la manière de les transformer. Les signaux peuvent être par exemple des images ou des enregistrements sonores.

Considérons un système  $H$  transformant un signal d’entrée  $e$  en signal de sortie  $s = H(e)$ . Des propriétés naturelles d’un tel système sont :

- $H$  est linéaire : si  $e_1$  et  $e_2$  sont des signaux et  $\alpha \in \mathbb{R}$ , alors  $H(e_1 + \alpha e_2) = H(e_1) + \alpha H(e_2)$  ;
- $H$  est continu
- $H$  est invariant par translation : si  $\forall t, H(e(t)) = s(t)$ , alors  $\forall t, H(e(t - \tau)) = s(t - \tau)$

On parle de système (ou filtre) linéaire, continu, et invariant. Ce sont des propriétés naturelles : la linéarité traduit que la réponse à deux signaux superposés est la superposition des réponses des signaux considérés de manière indépendante ; la continuité implique qu’une petite perturbation du signal en entrée induira une petite perturbation du signal en sortie ; l’invariance par translation implique que lorsqu’on traite un son, l’origine du temps n’importe pas, ou lorsqu’on traite une image, un « objet » est transformé de la même manière s’il est en haut à gauche ou en bas à droite de l’image.

On démontre alors que pour tout filtre vérifiant ces trois propriétés, il existe une fonction  $h$  (en fait, une distribution au sens de Laurent Schwartz. . .) telle que pour tout signal en

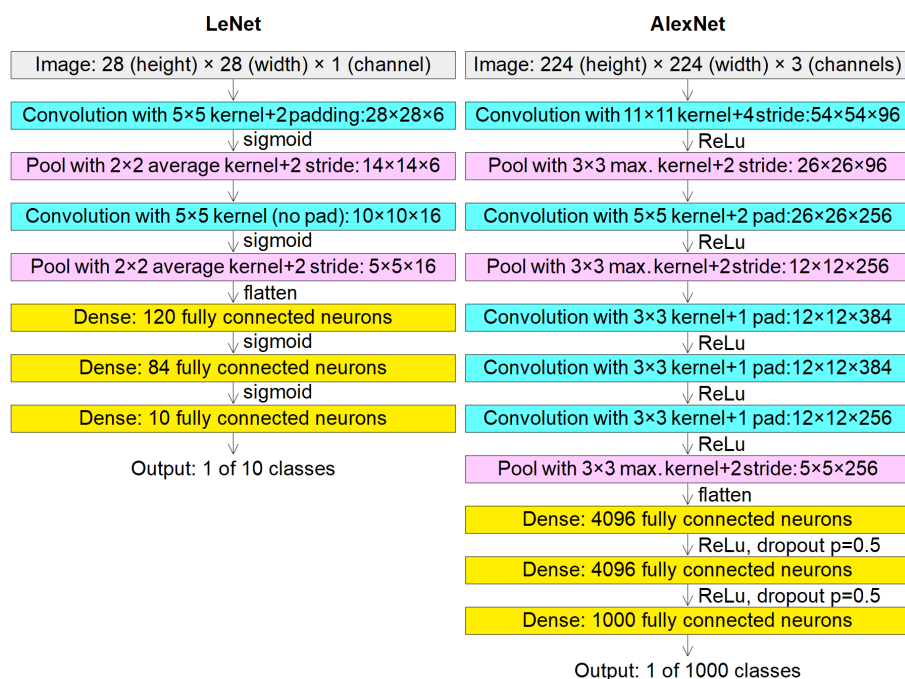


FIGURE 10.1 – *LeNet (à gauche) et AlexNet (à droite). Ces réseaux permettent de résoudre des problèmes de classification : 10 classes pour LeNet et 1000 classes pour AlexNet. LeNet admet en entrée des images en niveaux de gris de taille 32 × 32 pixels (28 × 28 indiqué ici à cause de la gestion des bords par les convolutions). Ce réseau est généralement utilisé pour la reconnaissance de chiffres manuscrits. AlexNet admet en entrée des images couleur de taille 224 × 224 pixels. LeNet a de l'ordre de 61 000 paramètres, et AlexNet 62 millions. Source de l'illustration : By Cmglee - Own work, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=104937230>*

entrée  $e$ ,

$$H(e) = h * s$$

où  $*$  désigne le produit de convolution. Rappelons que si  $h$  et  $s$  sont deux fonctions réelles intégrables, la convolution  $h * s$  est définie par :

$$\forall t \in \mathbb{R}, h * s(t) = \int_{\mathbb{R}} h(t-u)s(u) du = \int_{\mathbb{R}} h(u)s(t-u) du$$

Tout filtre s'exprime donc comme un produit de convolution et est caractérisé par une fonction  $h$  appelée « réponse impulsionnelle », ou noyau de convolution. On parle de réponse impulsionnelle car, dans le cadre de la théorie des distributions, la convolution de  $h$  avec la distribution de Dirac (une impulsion) est  $h$  elle-même.

Les convolutions apparaissent comme l'outil de base du traitement du signal. Lorsqu'on souhaite construire des réseaux de neurones traitant des signaux comme des images ou des sons, il semble donc naturel de s'appuyer sur les convolutions.

On s'intéresse en fait aux signaux numériques, qui sont discrets : ce sont des vecteurs  $(s_n)_n$ , indexés sur un espace discret plutôt que sur un espace continu comme la fonction  $s$  précédente.

On définit la convolution discrète de réponse impulsionnelle  $h$  comme :

$$\forall n \in \mathbb{Z}, h * s(n) = \sum_{k \in \mathbb{Z}} h(k)s(n - k)$$

En pratique, le noyau  $h$  est non nul pour un nombre fini de valeurs, la somme précédente ne concerne donc qu'un nombre fini de termes.

Dans le cas où le signal est une image, la convolution s'écrit comme une somme double :

$$\forall (m, n) \in \mathbb{Z}^2, h * s(m, n) = \sum_{(k, l) \in \mathbb{Z}^2} h(k, l)s(m - k, n - l)$$

L'opération de convolution étant linéaire, elle peut très bien être réalisée par une couche de neurones : les neurones d'entrées lisent le signal (fini), la couche cachée faite du même nombre de neurones calcule les  $h * s$ , mais pour ce faire les neurones cachés ne sont reliés qu'aux neurones d'entrée jouant un rôle dans la somme, et les poids des liaisons sont les coefficients du noyau. On voit que les poids des liaisons en entrée des neurones cachés sont les mêmes d'un neurone caché à l'autre : ce sont les coefficients du noyau de convolution.

Par exemple, pour réaliser la convolution d'un signal  $s(1), s(2), \dots, s(n)$  avec un noyau dont la taille de support est 3 :

$$\forall i \in \{1, 2, \dots, n\}, h * s(n) = \sum_{k \in \{-1, 0, 1\}} h(k)s(n - k)$$

donc le neurone caché d'indice  $n$  est relié aux neurones d'entrée d'indices  $n - 1, n, n + 1$  avec des poids respectifs  $h(1), h(0), h(-1)$ . Tous les neurones sont traités de la même manière : le noyau ne dépend pas de  $n$ . Dans cet exemple, les transitions entre les deux couches ne dépendent que de trois paramètres. Notons qu'il faudrait traiter les « bords » du signal  $s$  de manière particulière afin de donner un sens plus précis à la formule.

La figure 10.2 montre un exemple de l'effet de plusieurs filtres de convolution sur une image.

Comme en traitement du signal classique, on peut définir un « banc de filtres », qui sépare le signal en plusieurs composantes, chacune étant obtenue par un filtre autonome. Ici, le banc de filtre est implanté par des couches de convolution indépendantes utilisées sur la même entrée.

En toute généralité, l'entrée est d'« épaisseur »  $d$ . Par exemple  $d = 3$  pour une image couleur, car chaque pixel est représenté par un triple de composantes (rouge,vert,bleu). Un noyau de convolution sur une telle entrée est également d'« épaisseur »  $d$ .

La figure 10.3 représente une couche convolutive.

La sortie de chaque couche de convolution est rendue non-linéaire par une fonction d'activation. Afin d'assurer une certaine complexité au réseau, les couches de convolution peuvent se succéder. Si une première couche est formée de  $k$  filtres (elle représente un banc de filtres, comme précédemment évoqué), les noyaux de la seconde couche agiront sur toute



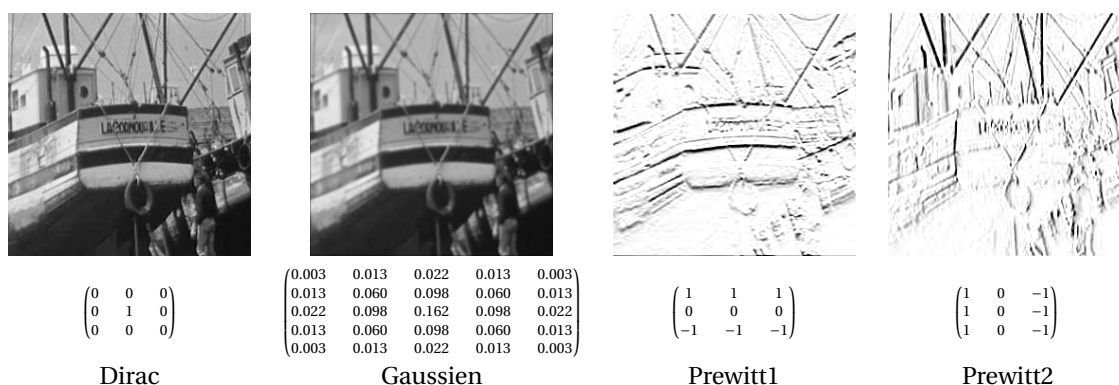


FIGURE 10.2 – Effet de différents filtres sur une image (en haut) et noyau de convolution (en bas). De gauche à droite : image originale (invariante par le filtre de Dirac), flou gaussien, filtre de Prewitt selon l'axe vertical, filtre de Prewitt selon l'axe horizontal. Le filtre gaussien (ici, support de taille  $5 \times 5$  et  $\sigma = 1$  pixel) rend l'image un peu floue, et les filtres de Prewitt mettent en évidence les contours horizontaux et verticaux (dans ces deux dernières images, par convention le noir représente une valeur élevée). Pour pouvoir calculer les convolutions sur les bords de l'image, on considère les niveaux de gris des pixels extérieurs à l'image comme égaux à 0, ce qui explique les effets de bords visibles.

la pile des  $k$  sorties. De manière à éviter le problème d'« évanouissement du gradient », on utilise une fonction d'activation ReLU qui n'est pas bornée plutôt qu'une sigmoïde, comme nous l'avons évoqué au paragraphe 9.4.5 du chapitre précédent. C'est le sens de “*we used non-saturating neurons*” dans la description du réseau AlexNet.

Les couches de convolution peuvent donc être vues comme des couches de neurones classiques auxquelles on impose des contraintes particulières sur les poids. Il se trouve que l'algorithme de rétropropagation s'adapte (la démonstration est tout de même un peu technique), et qu'on peut entraîner un réseau convolutif de manière similaire à ce qu'on a discuté au paragraphe 9.4 du chapitre 9. Plutôt qu'utiliser des filtres prédéfinis dédiés à des tâches précises comme en traitement du signal « classique », tel qu'illustré par la figure 10.2, l'apprentissage permet d'adapter les filtres en optimisant leurs paramètres. Cette approche est donc bien plus flexible que l'approche « traitement du signal » historique dans laquelle le spécialiste construisait des filtres ad-hoc.

### 10.2.2 Pooling

Pour limiter les coefficients à estimer et le temps de calcul, le support des noyaux de convolution est de taille petite. Par exemple, pour traiter des images on peut utiliser des noyaux de taille  $3 \times 3$  ou  $5 \times 5$  pixels (ou  $3 \times 3 \times 3$  ou  $5 \times 5 \times 3$  pour des images à trois canaux de couleur). Cependant, il semble légitime de vouloir intégrer des informations distantes de plus de trois ou cinq pixels. Une possibilité est de faire se succéder les convolutions. Une

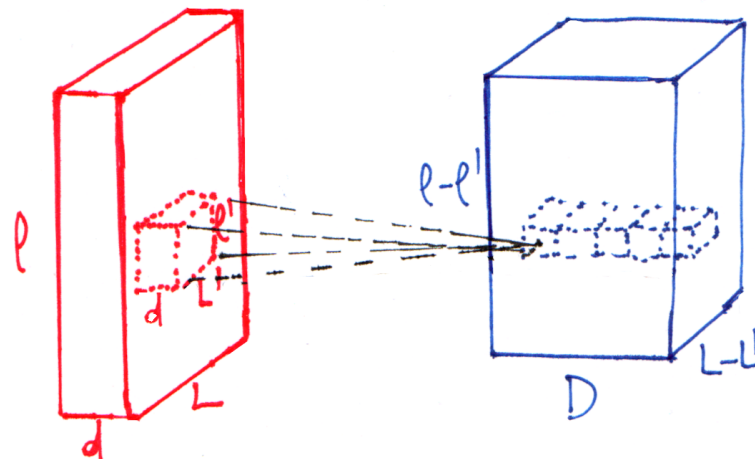


FIGURE 10.3 – Représentation d'une couche convolutive : entrée de la couche à gauche, sortie à droite. On suppose que l'entrée est formée d'un champ de dimension  $l \times L \times d$ . C'est le cas des signaux bidimensionnels comme les images, pour lesquelles  $d = 3$  si l'image a trois canaux de couleur. Un signal unidimensionnel serait représenté par un champ de dimension  $l \times 1 \times d$ , avec  $d = 1$  si le signal est univarié comme une onde sonore par exemple. Un filtre de support de taille  $l' \times L' \times d$  donne une valeur stockée dans un champ de dimension  $(l - l') \times (L - L') \times 1$  (sauf à traiter de manière particulière les bords, par exemple en mettant à zéro les valeurs manquantes pour obtenir une dimension de  $l \times L$ ), comme représenté par les pointillés noirs dans la figure. Ici, la couche convolutive implante un banc de  $D = 5$  filtres, la sortie de cette couche convolutive est donc un champ de dimension  $(l - l') \times (L - L') \times D$ . Cette couche dépend de  $D(l' L' d + 1)$  paramètres (« +1 » pour le biais). Elle est équivalente à une couche de réseau de neurones à propagation avant dans laquelle les poids sont partagés entre certaines connexions de manière à implanter des convolutions. Pour des raisons de lisibilité, on représente une couche convolutive par des blocs comme ici, plutôt qu'en disposant les neurones en colonne comme au chapitre 9.

autre possibilité est de réduire la taille des couches représentant le signal convolué par sous-échantillonnage, par exemple en prenant le plus grand élément parmi deux éléments consécutifs (ou parmi un bloc de  $2 \times 2$  pixels dans le cas des images). Un noyau opérant ensuite sur une couche sous-échantillonnée par un facteur 2 aurait un effet semblable à un noyau de support de taille double sur la couche originale. L'avantage est de limiter le nombre de paramètre des noyaux successifs.

On appelle couche de *pooling* (« mise en commun ») les couches du réseau effectuant une telle opération. Le *max-pooling* illustré par la figure 10.4 introduit une nouvelle non-linéarité. Notons que les couches de *pooling* n'introduisent pas de nouveaux paramètres à apprendre.

### 10.2.3 Dropout

Pour éviter le sur-apprentissage, le *dropout* est couramment utilisé dans les couches complètement connectées du réseau (généralement les couches près de la sortie, comme on le

|      |      |      |      |   |     |      |
|------|------|------|------|---|-----|------|
| 5.5  | 3.9  | -5.1 | -4.0 | → | 5.5 | -1.1 |
| 1.1  | 2.4  | -6.1 | -1.1 |   | 3.9 | 3.1  |
| -4.2 | 2.0  | 3.1  | 1.2  |   |     |      |
| 3.9  | -0.4 | -0.3 | -3.1 |   |     |      |

FIGURE 10.4 – Exemple de pooling par sous-échantillonnage d'un facteur 2 : on passe ici d'une couche de dimension  $4 \times 4$  à une couche de dimension  $2 \times 2$ . Chaque bloc de dimension  $2 \times 2$  est remplacé par une valeur unique, égale au maximum des valeurs présentes dans le cas du max-pooling. On peut envisager d'autres formes de pooling, par exemple un average-pooling où on remplace un bloc par sa valeur moyenne.

verra dans l'exemple de VGG16, figure 10.5). Cette procédure consiste à « déconnecter » aléatoirement certains neurones d'une couche avec une probabilité  $p$  (par exemple  $p = 0.5$ ), pendant une itération de l'apprentissage par gradient stochastique. Les poids relatifs à ces neurones ne sont donc pas mis à jour. Cela a pour effet de limiter le sur-apprentissage (et joue donc un rôle de régularisation), en particulier en évitant d'apprendre des caractéristiques communes à un groupe de neurones. En effet, la probabilité qu'un groupe donné de  $k$  neurones soit actif dans une itération de l'apprentissage est  $(1 - p)^k$ , ce qui est très faible.

Pour donner une intuition dans un problème de classification de chiens et chats où les images de chiens montrent systématiquement un ciel bleu et celles de chats un gazon vert, le *dropout* devrait permettre d'éviter que toute image montrant un ciel bleu (respectivement un gazon) soit reconnue comme un chien (respectivement un chat), en forçant l'apprentissage à également prendre en compte d'autres caractéristiques.

Pour la prédiction, tous les neurones sont utilisés. Le *dropout* n'est employé que pour la phase d'apprentissage.

#### 10.2.4 Augmentation des données

Si l'apprentissage profond s'appuie souvent sur de très grandes bases d'apprentissage pour permettre l'estimation du grand nombre de paramètres des modèles, on peut également augmenter cette base en simulant de nouvelles observations à l'aide de transformations prédéfinies. Par exemple, si on cherche à reconnaître des véhicules dans des images, et qu'on dispose uniquement d'exemples d'images dans lesquelles le sol est horizontal, on peut enrichir la base d'images en générant par des rotations d'angles variés des images synthétiques dans lesquelles le sol est penché. De cette manière, on espère parvenir à reconnaître les véhicules lorsque les images de test ne présentent pas un sol horizontal.

L'enrichissement par génération de données synthétiques peut être fait « au vol » pendant l'apprentissage en générant aléatoirement les paramètres des transformations appliquées à chaque itération. De cette manière, ce ne sont jamais les mêmes observations qui sont présentées au réseau pendant l'apprentissage, ce qui permet de se prémunir du sur-apprentissage.

### 10.2.5 Nombre de paramètres à entraîner

Les paramètres à entraîner sont ceux des couches de convolution, ainsi que ceux des parties denses, comme l'illustrent les exemples de LeNet et d'AlexNet (Figure 10.1). Considérons le réseau LeNet :

- La première couche de convolution est constituée d'un banc de 6 filtres de taille  $5 \times 5$ . Le nombre de paramètres est donc :  $(5 \times 5 + 1) \times 6 = 156$  (« +1 » pour le terme de biais dans chaque filtre).
- La couche de *pooling* suivante n'introduit pas de nouveaux paramètres.
- La seconde couche de convolution est constituée de 16 filtres  $5 \times 5 \times 6$  («  $\times 6$  » car la couche précédente est d'épaisseur 6). Le nombre de paramètres est :  $(5 \times 5 \times 6 + 1) \times 16 = 2416$ .
- La sortie de la seconde couche de *pooling* est de dimension  $5 \times 5 \times 16 = 400$ .
- Ces 400 neurones sont ensuite connectés de manière dense à 120 neurones, d'où  $(400 + 1) \times 120 = 48120$  paramètres.
- Ces 120 neurones sont connectés à 84 neurones, d'où  $(120 + 1) \times 84 = 10164$  paramètres.
- Ces 84 neurones sont connectés aux 10 neurones de sorties, d'où  $(84 + 1) \times 10 = 850$  paramètres.

Soit un total de  $156 + 2416 + 48120 + 10164 + 850 = 61706$  paramètres à entraîner.

## 10.3 L'exemple de VGG16

Un réseau convolutif typique est constitué de la succession de plusieurs blocs de convolution. Chaque bloc est constitué de plusieurs couches de convolution indépendantes (elles implantent différents filtres, appris lors de l'entraînement). L'information est ensuite passée au bloc suivant après *pooling*. Par conséquent, plus on avance dans le réseau, plus les convolutions intègrent de l'information distante dans l'image originale. Intuitivement, les couches de convolution proches de l'entrée apprennent des corrélations locales (pour reconnaître un visage on détecte d'abord le nez, les yeux, etc), et les couches de convolution finales apprennent des corrélations distantes (un visage est constitué d'un nez à peu près au milieu de deux yeux). Finalement, tout est intégré dans un réseau de neurones multicouches à propagation avant classique, avec des connexions entre tous les neurones appartenant à des couches successives, et une dernière couche de type SoftMax. Les dernières couches peuvent donc être vues comme un classifieur comme au Chapitre 9 opérant sur les informations données par les couches de convolution qui, pour leur part, extraient des images des caractéristiques pertinentes apprises sur la base.

La figure 10.5 montre l'exemple de l'architecture du réseau VGG16. Par comparaison à LeNet et AlexNet (figure 10.1), l'architecture est semblable, la différence tenant essentiellement dans le nombre de paramètres à apprendre. De nombreuses autres architectures ont été proposées pour le problème de la reconnaissance d'images, et de nombreuses autres sont proposées pour d'autres domaines applicatifs. En 2022, 8271 articles scientifiques déposés sur le

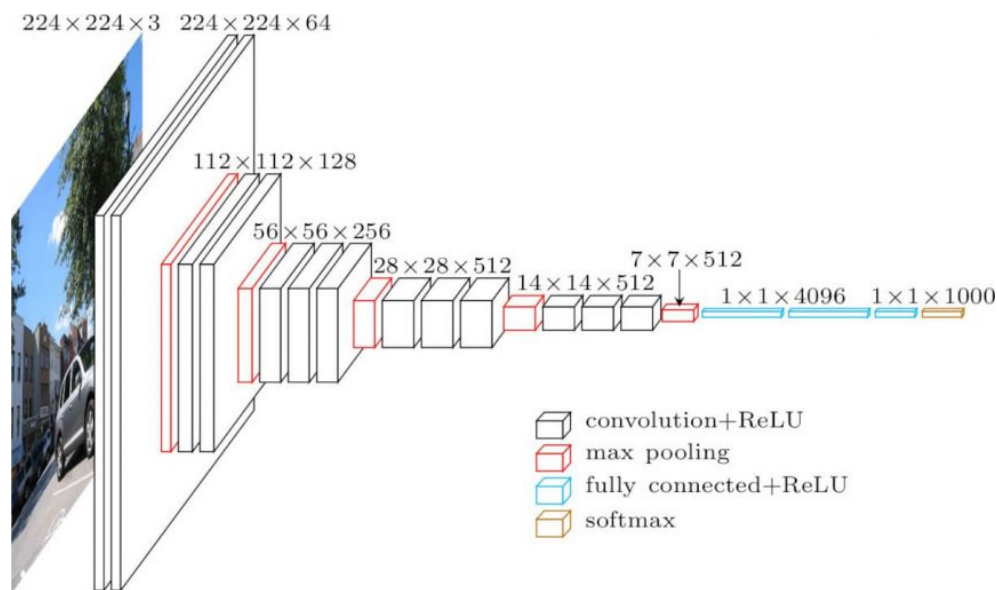


FIGURE 10.5 – Un exemple de réseau convolutif profond : VGG16 (K. Simonyan and A. Zisserman, 2014). L'entrée est constituée de  $224 \times 224 \times 3 = 150528$  neurones, ce qui correspond à une image couleur (trois canaux) de  $224 \times 224$  pixels. Le premier bloc de convolutions est constitué de 64 filtres indépendants de support de taille  $3 \times 3 \times 3$ , chacun étant suivi d'une activation ReLU. Un second bloc réalise la même opération. Ensuite, un seul pixel sur deux est gardé dans le pooling. Les blocs de convolution et de pooling se succèdent jusqu'à atteindre un réseau complètement connecté classique constitué de deux couches de 4096 neurones, une couche de 1000 neurones, puis une couche SoftMax de 1000 neurones (il s'agit d'un problème de classification à 1000 classes). Ce réseau est gouverné par 138 millions de paramètres. Par exemple les connexions entre l'entrée et la première couche de convolution nécessitent  $(3 \times 3 \times 3 + 1) \times 64 = 1792$  paramètres (64 filtres  $3 \times 3$  sur les 3 composantes RVB, et un biais par filtre), et entre la première et la deuxième couche de convolution  $(3 \times 3 \times 64 + 1) \times 64 = 36928$  paramètres. Source de l'illustration : <https://neurohive.io/en/popular-networks/vgg16/> où sont fournis les détails de l'architecture.

serveur de preprints Arxiv mentionnaient *deep learning* dans leur résumé...<sup>1</sup>

## 10.4 Apprentissage par transfert et réglage fin

Bien entendu, le grand nombre de paramètres à apprendre nécessite une grande base de données et des ressources de calcul conséquentes. Par exemple, VGG16 est construit à l'aide de 14 millions d'images de la base de données ImageNet et a nécessité en 2014 plusieurs semaines d'apprentissage. Il est à présent disponible sur Internet, comme de nombreux réseaux

1. 2022 : 7622, 2021 : 7210; 2020 : 6194; 2019 : 4150; 2018 : 2791; 2017 : 1360; 2016 : 500...

construits pour différents problèmes. Nous aimerions pouvoir réemployer ces réseaux pour faire de l'apprentissage profond sur nos propres problèmes, sans refaire un long apprentissage...

Cependant, que faire si on s'intéresse à un problème de reconnaissance d'images qui ne seraient pas issues d'ImageNet? Et si on cherche à identifier non pas 1000 catégories comme dans ImageNet mais 5 qui ne figureraient pas dans cette base d'images? Les couches de convolution apprennent des descripteurs des images, qui sont ensuite fournis en entrée à la partie *fully connected* supérieure du réseau. En supposant les images de la base ImageNet « pas trop éloignées » des images de notre problème, on peut imaginer réutiliser les descripteurs de VGG16 et donc utiliser les couches de convolution correspondantes. Ensuite, on change la couche SoftMax de sortie de manière à prédire 5 classes, on change éventuellement le nombre de neurones des couches *fully connected*, puis on entraîne uniquement les dernières couches du réseau qui servent à la classification. Alors qu'on ne pouvait pas entraîner intégralement un réseau comme VGG16 sur une petite base d'images, il est possible d'entraîner uniquement les dernières couches sur cette base, en gardant fixes les poids des couches convolutives. On pourrait d'ailleurs même envisager d'utiliser un autre classifieur que la partie *fully connected*, par exemple en classifiant les sorties de la partie convolutive à l'aide d'une SVM. Cette manière de procéder s'appelle « apprentissage par transfert » (*transfer learning*).

Une approche complémentaire est de poursuivre l'apprentissage d'un réseau « pré-appris » (*pre-trained network*) comme VGG16, avec les observations qui nous intéressent. En initialisant l'apprentissage avec les poids originaux du réseau et en utilisant un taux d'apprentissage faible, on ne fait que modifier légèrement les poids en les adaptant à nos observations. On parle de « réglage fin » du réseau (*fine tuning*).

## 10.5 Pour approfondir...

**Historique** Les deux articles suivants présentent un historique de la révolution *deep learning*:

D. Cardon, J.-Ph. Cointet, A. Mazières, *La revanche des neurones. L'invention des machines inductives et la controverse de l'intelligence artificielle*, Réseaux, vol. 5, no. 211, p. 173-220, 2018.

<https://www.cairn.info/revue-reseaux-2018-5-page-173.htm>

V. Lepetit, *Yann Le Cun, prix Turing 2018 : des idées profondes*, 1024 - Bulletin de la société informatique de France, no. 14, p. 99-102, 2019.

<https://www.societe-informatique-de-france.fr/bulletin/1024-numero-14/>

**Lecture** Nous n'avons fait qu'esquisser quelques éléments sur l'apprentissage profond à travers les réseaux convolutifs, et n'avons évoqué que l'application à la classification d'images. Par exemple, les approches modernes de l'apprentissage par réseaux de neurones permettent l'apprentissage non-supervisé (voir les auto-encodeurs) ou l'apprentissage de séries temporelles ou de données textuelles (voir les réseaux de neurones récurrents). Un point d'entrée

pour aller (beaucoup) plus loin est le livre :

I. Goodfellow, Y. Bengio, A. Courville, *Deep learning*, MIT Press, 2016.

Les lecteurs intéressés pourront également consulter les cours de Stéphane Mallat, titulaire de la chaire « sciences des données » au Collège de France (vidéos et textes sont disponibles) :

<https://www.college-de-france.fr/fr/chaire/stephane-mallat-sciences-des-donnees-chaire-status>

Enfin un livre gratuit très récent présente une excellente introduction à l'apprentissage profond, des architectures dépassant les réseaux convolutifs aux applications modernes :

F. Fleuret, *The little book of deep learning*, 2023.

<https://fleuret.org/francois/lbdl.html>

Ceux parmi vous souhaitant approfondir le sujet sont invités à consulter les deux livres cités en préface, disponibles à la bibliothèque.





## Quelques résultats utiles

### A.1 Inégalités de Hoeffding

Soient  $X_1, X_2, \dots, X_n$   $n$  variables aléatoires indépendantes telles que pour tout  $i$ ,  $X_i$  prend ses valeurs dans un intervalle  $[a_i, b_i]$ . Notons

$$\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i$$

la moyenne empirique.

**Proposition A.1** Pour tout  $t \geq 0$ , les inégalités de Hoeffding (1963) s'écrivent :

$$\Pr(\bar{X} - E(\bar{X}) \geq t) \leq \exp\left(\frac{-2n^2 t^2}{\sum_{i=1}^n (b_i - a_i)^2}\right)$$

et :

$$\Pr\left(\left|\bar{X} - E(\bar{X})\right| \geq t\right) \leq 2 \exp\left(\frac{-2n^2 t^2}{\sum_{i=1}^n (b_i - a_i)^2}\right)$$

Une démonstration se trouve ici :

[https://fr.wikipedia.org/wiki/In%C3%A9galit%C3%A9\\_de\\_Hoeffding](https://fr.wikipedia.org/wiki/In%C3%A9galit%C3%A9_de_Hoeffding)

### A.2 Dérivation des fonctions composées

Soit  $\phi$  une fonction différentiable de  $\mathbb{R}^n$  dans  $\mathbb{R}^m$  ( $n, m$  entiers non nuls). La matrice jacobienne de  $\phi$  en  $\mathbf{x} \in \mathbb{R}^n$  est la matrice suivante, à  $m$  lignes et  $n$  colonnes :

$$J_{\phi}(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial f_m}{\partial x_n}(\mathbf{x}) \end{pmatrix}$$

**Proposition A.2** On considère deux fonctions différentiables :  $g$  de  $\mathbb{R}^p$  dans  $\mathbb{R}^q$  et  $f$  de  $\mathbb{R}^q$  dans  $\mathbb{R}^r$  (pour  $p, q$  et  $r$  des entiers non nuls). Alors la composée  $h = f \circ g$  de  $\mathbb{R}^p$  dans  $\mathbb{R}^r$  est différentiable et sa jacobienne est la matrice à  $r$  lignes et  $p$  colonnes donnée par :

$$J_h(\mathbf{x}) = J_f(g(\mathbf{x})) J_g(\mathbf{x})$$

Il s'agit de la formule de dérivation des fonctions composées (*chain rule*). Le cas particulier  $p = q = r = 1$  donne la formule bien connue :  $h'(x) = g'(x)f'(g(x))$ .

Dans le chapitre 9 (section 9.4.3), nous avons besoin de ce résultat dans deux cas particuliers :

1. Cas particulier de deux fonctions différentiables :  $g$  de  $\mathbb{R}^p$  dans  $\mathbb{R}^q$  et  $f$  de  $\mathbb{R}^q$  dans  $\mathbb{R}$  (ici  $r = 1$ ). On souhaite calculer les dérivées partielles de la fonction composée  $h = f \circ g$  de  $\mathbb{R}^p$  dans  $\mathbb{R}$ .

Si  $\mathbf{x} = (x_1, \dots, x_p) \in \mathbb{R}^p$  et  $\mathbf{y} = g(\mathbf{x}) = (y_1, \dots, y_q) \in \mathbb{R}^q$ , alors la dérivée partielle de  $h$  par rapport à la  $i$ -ème composante de sa variable est donnée par l'expression suivante :

$$\forall i \in [1, p], \frac{\partial h}{\partial x_i}(\mathbf{x}) = \sum_{j=1}^q \frac{\partial f}{\partial y_j}(\mathbf{y}) \frac{\partial g_j}{\partial x_i}(\mathbf{x})$$

où  $\frac{\partial f}{\partial y_j}$  désigne la dérivée partielle de  $f$  par rapport à sa  $j$ -ème composante, et  $\frac{\partial g_j}{\partial x_i}$  la dérivée partielle de  $g_j$ ,  $j$ -ème composante de  $g$ , par rapport à sa  $i$ -ème composante.

2. Cas  $q = r = 1$  :

$$\forall i \in [1, p], \frac{\partial h}{\partial x_i}(\mathbf{x}) = f'(\mathbf{y}) \frac{\partial g}{\partial x_i}(\mathbf{x})$$

où  $f'$  désigne la dérivée de la fonction réelle  $f$ .

## A.3 Matrices symétriques, classification des quadriques et coniques

### A.3.1 Rappels

Une matrice carrée  $A$  à coefficients réels est dite symétrique si elle est égale à sa transposée :  $A = A^T$ .

Une matrice carrée  $P$  à coefficients réels est dite orthogonale si son inverse est égal à sa transposée :  $PP^T = P^T P = \text{Id}$ . Autrement dit, les vecteurs colonnes et les vecteurs lignes de  $P$  forment une base orthonormée de  $\mathbb{R}^n$ .

**Proposition A.3** Une matrice  $A$  symétrique d'ordre  $n$  est diagonalisable à l'aide de matrices de passage orthogonale, c'est-à-dire qu'il existe des matrices  $D$  et  $P$  d'ordre  $n$  telles que :

$$A = PDP^T$$

où  $D$  est diagonale et  $P$  est orthogonale.

Les coefficients de  $D$  sont les valeurs propres de  $A$  et les colonnes de  $P$  sont les vecteurs propres de  $A$ .

$A$  admet donc  $n$  valeurs propres (distinctes ou non).

Une matrice symétrique  $A$  est dite positive si  $\forall \mathbf{x} \in \mathbb{R}^n, \mathbf{x}^T A \mathbf{x} \geq 0$ .

Elle est dite symétrique définie positive si  $\forall \mathbf{x} \in \mathbb{R}^n \setminus \{0\}, \mathbf{x}^T A \mathbf{x} > 0$ .

**Proposition A.4** *Les valeurs propres d'une matrice symétrique positive sont positives ou nulles.*

*Les valeurs propres d'une matrice symétrique définie positive sont strictement positives. En particulier, cette matrice est inversible.*

*Démonstration*

Si  $A$  est une matrice symétrique positive, soient  $\lambda$  une valeur propre de  $A$  et  $\mathbf{x}$  un vecteur propre associé. On a  $\mathbf{x}^T A \mathbf{x} = \lambda |\mathbf{x}|^2$  et  $\mathbf{x}^T A \mathbf{x} \geq 0$ , donc  $\lambda \geq 0$ . Le cas défini se traite de la même manière avec des inégalités strictes.  $\square$

**Proposition A.5** *Si  $A$  est une matrice symétrique définie positive, alors  $A^{-1}$  aussi.*

*Démonstration*

Si  $A$  est symétrique définie positive, ses valeurs propres sont strictement positives donc  $A$  est inversible.  $(A^{-1})^T = (A^T)^{-1} = A^{-1}$  donc l'inverse de  $A$  est symétrique. Soient  $P$  orthogonale et  $D$  diagonale telles que  $A = P D P^T$ . Alors  $A^{-1} = P D^{-1} P^T$  car  $P^{-1} = P^T$ . Les éléments diagonaux de  $D^{-1}$  sont comme ceux de  $D$  strictement positifs, donc pour tout  $\mathbf{x}$  non nul :  $\mathbf{x}^T A^{-1} \mathbf{x} = (P^T \mathbf{x})^T D^{-1} P^T \mathbf{x} > 0$ .  $\square$

### A.3.2 Classification des quadriques et coniques

Une quadrique de l'espace affine  $\mathbb{R}^3$  est l'ensemble des points de coordonnées  $(x_1, x_2, x_3)$  tels que :

$$a_{11}x_1^2 + a_{22}x_2^2 + a_{33}x_3^2 + a_{12}x_1x_2 + a_{13}x_1x_3 + a_{23}x_2x_3 + b_1x_1 + b_2x_2 + b_3x_3 + c = 0$$

où les  $a_{ij}$  ( $i, j \in \{1, 2, 3\}$ ),  $b_i$  ( $i \in \{1, 2, 3\}$ ) et  $c$  sont des coefficients réels.

De manière équivalente, si  $a_{11}$ ,  $a_{22}$  et  $a_{33}$  ne sont pas nuls :

$$\begin{aligned} & a_{11} \left( x_1 + \frac{b_1}{2a_{11}} \right)^2 + a_{22} \left( x_2 + \frac{b_2}{2a_{22}} \right)^2 + a_{33} \left( x_3 + \frac{b_3}{2a_{33}} \right)^2 \\ & + a_{12} \left( x_1 + \frac{b_1}{2a_{11}} \right) \left( x_2 + \frac{b_2}{2a_{22}} \right) + a_{13} \left( x_1 + \frac{b_1}{2a_{11}} \right) \left( x_3 + \frac{b_3}{2a_{33}} \right) + a_{23} \left( x_2 + \frac{b_2}{2a_{22}} \right) \left( x_3 + \frac{b_3}{2a_{33}} \right) \\ & + c - \frac{b_1^2}{4a_{11}} - \frac{b_2^2}{4a_{22}} - \frac{b_3^2}{4a_{33}} - \frac{a_{12}b_1b_2}{4a_{11}a_{22}} - \frac{a_{13}b_1b_3}{4a_{11}a_{33}} - \frac{a_{23}b_2b_3}{4a_{22}a_{33}} = 0 \end{aligned}$$

qui est la forme canonique de la quadrique, dans laquelle on a fait apparaître le centre de la conique de coordonnées :

$$\left( -\frac{b_1}{2a_{11}} \quad -\frac{b_2}{2a_{22}} \quad -\frac{b_3}{2a_{33}} \right)$$

Le cas où l'un des  $a_{ii}$  est nul se traite de manière similaire.

Par changement d'origine, on se ramène ainsi à l'étude de l'ensemble des  $\mathbf{x}$  de  $\mathbb{R}^3$  tels que :

$$\mathbf{x}^T A \mathbf{x} + \delta = 0$$

où  $\delta$  est un coefficient réel et  $A$  est la matrice symétrique :

$$A = \begin{pmatrix} a_{11} & a_{12}/2 & a_{13}/2 \\ a_{12}/2 & a_{22} & a_{23}/2 \\ a_{13}/2 & a_{23}/2 & a_{33} \end{pmatrix}$$

Ainsi, toute quadrique est associée à une matrice symétrique.

Une matrice symétrique étant diagonalisable dans la base orthonormée de ses vecteurs propres, en écrivant  $A = PDP^T$ , et en notant  $\mathbf{y} = P^T \mathbf{x}$  et  $s_1, s_2, s_3$  les éléments diagonaux de  $D$  (ce sont les valeurs propres de  $A$ ), on est ramené à l'étude de :

$$s_1 y_1^2 + s_2 y_2^2 + s_3 y_3^2 = \delta$$

qui est l'équation réduite de la quadrique.

Discuter la nature des quadriques selon les valeurs propres  $s_1, s_2, s_3$  de la matrice symétrique  $A$  est ce qu'on appelle la classification des quadriques. Nous vous renvoyons vers la littérature dédiée.

Le petit calcul que nous venons de faire s'adapte aisément selon la dimension de l'espace affine. En dimension 2, on parle de conique : de la même manière que précédemment, on classe les coniques selon les valeurs propres de  $A$ .

En dimension 2 (resp. 3), si  $A$  est symétrique définie positive, alors les deux (resp. trois) valeurs propres sont strictement positives. Si  $\delta$  est négatif, l'équation réduite nous indique que la surface définie est une ellipse (resp. un ellipsoïde).

### A.3.3 Propriétés utiles dans le cours

Ce cours utilise des résultats sur les matrices symétriques dans plusieurs cas.

#### A.3.3.1 Lignes de niveau des densités de probabilité gaussiennes

Les lignes de niveaux d'une densité de probabilité gaussienne sont des ellipses en dimension  $d = 2$  ou des ellipsoïdes en dimension  $d = 3$ , centrées en la moyenne de la densité. En effet, une densité gaussienne de moyenne  $\boldsymbol{\mu}$  et de matrice de covariance  $\Sigma$  s'écrit :

$$f_{\boldsymbol{\mu}, \Sigma}(\mathbf{x}) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})}$$

Une ligne de niveau est formée des  $\mathbf{x}$  tels que  $f_{\mu, \Sigma}(\mathbf{x}) = c$  où  $c$  est une constante positive, donc (en passant au logarithme) ses points vérifient :

$$(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}) = K$$

où  $K$  est un paramètre réel.

Une matrice de covariance est symétrique définie positive par définition, donc son inverse également. Les lignes de niveau des densités de probabilité gaussiennes sont donc bien des ellipses ou ellipsoïdes centrés en  $\boldsymbol{\mu}$  lorsque  $K > 0$ , ou l'ensemble vide lorsque  $K < 0$ . La discussion précédente nous indique que les vecteurs propres de  $\Sigma$  portent les axes de l'ellipse ou ellipsoïde, et que la longueur de ces axes est proportionnelle aux valeurs propres de  $\Sigma$ . Dans le cas où la matrice de covariance est diagonale (les composantes du vecteur gaussien étant alors décorréélées), la longueur des axes est proportionnelle à l'écart-type de la composante. Les ellipses ou ellipsoïdes correspondant aux lignes de niveau montrent bien la répartition des réalisations du vecteur gaussien.

### A.3.3.2 Nature des points critiques d'une fonction de plusieurs variables

Le développement limité à l'ordre 2 d'une fonction  $f$  de classe  $C^2$  d'un ouvert de  $\mathbb{R}^n$  dans  $\mathbb{R}$  s'écrit en  $\mathbf{x}_0$  :

$$f(\mathbf{x}) =_{\mathbf{x}_0} f(\mathbf{x}_0) + \nabla f_{\mathbf{x}_0}^T \cdot (\mathbf{x} - \mathbf{x}_0) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_0)^T Hf_{\mathbf{x}_0} (\mathbf{x} - \mathbf{x}_0) + o(|\mathbf{x} - \mathbf{x}_0|^2)$$

où  $\nabla f_{\mathbf{x}_0}$  est le vecteur gradient et  $Hf_{\mathbf{x}_0}$  est la matrice hessienne en  $\mathbf{x}_0$ , c'est-à-dire :

$$\nabla f_{\mathbf{x}_0} = \left( \frac{\partial f}{\partial x_i}(\mathbf{x}_0) \right)_{1 \leq i \leq n}$$

et :

$$Hf_{\mathbf{x}_0} = \left( \frac{\partial^2 f}{\partial x_i \partial x_j}(\mathbf{x}_0) \right)_{1 \leq i, j \leq n}$$

Comme  $f$  est  $C^2$ , la matrice hessienne  $Hf_{\mathbf{x}_0}$  est symétrique (théorème de Schwarz).

La matrice hessienne nous permet de déterminer la nature des points où le gradient de  $f$  s'annule (appelés points critiques).

Au voisinage de  $\mathbf{x}_0$  tel que  $\nabla f_{\mathbf{x}_0} = 0$  :

$$f(\mathbf{x}) \simeq f(\mathbf{x}_0) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_0)^T Hf_{\mathbf{x}_0} (\mathbf{x} - \mathbf{x}_0)$$

Ainsi,

- si  $Hf_{\mathbf{x}_0}$  est symétrique positive,  $f(\mathbf{x}) \geq f(\mathbf{x}_0)$  pour  $\mathbf{x}$  au voisinage de  $\mathbf{x}_0$  et  $f(\mathbf{x}_0)$  est un minimum local,
- si  $-Hf_{\mathbf{x}_0}$  est symétrique positive (dans ce cas la matrice  $Hf_{\mathbf{x}_0}$  est dite symétrique négative),  $f(\mathbf{x}) \leq f(\mathbf{x}_0)$  pour  $\mathbf{x}$  au voisinage de  $\mathbf{x}_0$  et  $f(\mathbf{x}_0)$  est un maximum local,
- si certaines valeurs propres de la hessienne sont négatives et d'autres positives, alors pour tout vecteur propre  $\tilde{\mathbf{x}}$  associé à une valeur propre  $\sigma$ ,  $f(\mathbf{x}_0 + \lambda \tilde{\mathbf{x}}) \simeq f(\mathbf{x}_0) + \frac{\lambda^2 \sigma}{2} |\tilde{\mathbf{x}}|^2$  pour une valeur de  $\lambda$  « assez petite ». Dans ce cas,  $f(\mathbf{x}_0 + \lambda \tilde{\mathbf{x}}) \leq f(\mathbf{x}_0)$  si  $\sigma < 0$  et  $f(\mathbf{x}_0 + \lambda \tilde{\mathbf{x}}) \geq f(\mathbf{x}_0)$  si  $\sigma > 0$  : selon la direction  $\tilde{\mathbf{x}}$  que l'on suit au voisinage de  $\mathbf{x}_0$ , la valeur de  $f$  décroît ou croît. On dit que  $\mathbf{x}_0$  est un point col (ou point selle, *saddle point*).



## Rappels d'optimisation

### B.1 Éléments d'optimisation convexe

À plusieurs reprises dans ce cours nous avons à optimiser des fonctions convexes. Rappelons quelques propriétés de ces fonctions.

#### B.1.1 Définition

Résoudre un problème d'optimisation convexe consiste à chercher le minimum d'une fonction  $f$  convexe sur un ensemble (domaine)  $E \subset \mathbb{R}^d$  convexe.

On rappelle qu'un ensemble  $E$  est convexe si pour toute paire de points  $(\mathbf{x}_1, \mathbf{x}_2) \in E \times E$ , le segment  $[\mathbf{x}_1, \mathbf{x}_2]$  est inclus dans  $E$ , soit :

$$\forall (\mathbf{x}_1, \mathbf{x}_2) \in E, \forall \alpha \in [0, 1], \alpha \mathbf{x}_1 + (1 - \alpha) \mathbf{x}_2 \in E$$

Une fonction à valeurs réelles est convexe sur  $E$  convexe si pour toute paire de points  $(\mathbf{x}_1, \mathbf{x}_2) \in E \times E$ , la corde joignant  $f(\mathbf{x}_1)$  et  $f(\mathbf{x}_2)$  est au dessus du graphe de  $f$  entre  $\mathbf{x}_1$  et  $\mathbf{x}_2$ , soit :

$$\forall (\mathbf{x}_1, \mathbf{x}_2) \in E, \forall \alpha \in [0, 1], f(\alpha \mathbf{x}_1 + (1 - \alpha) \mathbf{x}_2) \leq \alpha f(\mathbf{x}_1) + (1 - \alpha) f(\mathbf{x}_2)$$

Lorsque l'inégalité est stricte avec  $\mathbf{x}_1 \neq \mathbf{x}_2$  et  $\alpha \in ]0, 1[$ ,  $f$  est dite strictement convexe.

#### B.1.2 Propriétés

La convexité implique l'inégalité suivante (appelée inégalité de Jensen), souvent très utile.

**Proposition B.1** Soient  $f$  est une fonction convexe sur un domaine  $E$  convexe,  $\mathbf{x}_1, \dots, \mathbf{x}_n \in E$ ,  $\alpha_1, \dots, \alpha_n \geq 0$  tels que  $\sum_i \alpha_i = 1$  alors :

$$f\left(\sum_{i=1}^n \alpha_i \mathbf{x}_i\right) \leq \sum_{i=1}^n \alpha_i f(\mathbf{x}_i)$$

En d'autres termes, l'image par  $f$  d'une moyenne pondérée d'un ensemble de points est inférieure à la même moyenne pondérée des images de ces points.

Si  $f$  est de classe  $C^1$ , on dispose de la caractérisation suivante :

**Proposition B.2**  $f$  est convexe sur  $E$  si et seulement si

$$\forall (\mathbf{x}_1, \mathbf{x}_2) \in E \times E, f(\mathbf{x}_1) \geq f(\mathbf{x}_2) + (\mathbf{x}_1 - \mathbf{x}_2) \cdot \nabla f(\mathbf{x}_2)$$

Ceci signifie que le graphe d'une fonction convexe est au dessus de tout hyperplan tangent à ce graphe.

Si  $f$  est de classe  $C^2$ , on dispose de la caractérisation suivante :

**Proposition B.3**  $f$  est convexe sur  $E$  si et seulement si la matrice hessienne de  $f$  (c'est-à-dire la matrice des dérivées secondes de  $f$ ) en tout  $\mathbf{x} \in E$  est symétrique positive.

Les fonctions convexes jouent un rôle particulier en optimisation à cause de la proposition suivante :

**Proposition B.4** Si  $f$  est une fonction convexe sur  $E$  convexe, alors :

- tout minimum local est global :
- si, de plus,  $f$  est de classe  $C^1$  et  $E$  est ouvert, alors  $\mathbf{x}^* \in \operatorname{argmin}_{\mathbf{x} \in E} f$  si et seulement si  $\nabla f(\mathbf{x}^*) = 0$ .

*Démonstration*

Si  $\mathbf{x}^*$  est minimum local, soit  $\mathbf{y} \in E$ . Par définition de la convexité, pour tout  $\alpha \in [0, 1]$  et  $\mathbf{y} \in E$ ,

$$f(\alpha \mathbf{y} + (1 - \alpha) \mathbf{x}^*) \leq \alpha f(\mathbf{y}) + (1 - \alpha) f(\mathbf{x}^*)$$

soit :

$$f(\alpha \mathbf{y} + (1 - \alpha) \mathbf{x}^*) - f(\mathbf{x}^*) \leq \alpha (f(\mathbf{y}) - f(\mathbf{x}^*))$$

Comme  $\mathbf{x}^*$  est minimum local, pour  $\alpha$  assez petit le membre de gauche est positif, donc le membre de droite aussi, quel que soit  $\mathbf{y} \in E$ .  $\mathbf{x}^*$  est donc minimum global.

Supposons  $f$  de classe  $C^1$  et  $E$  ouvert. Si  $\mathbf{x}^*$  est minimum local, alors  $\nabla f(\mathbf{x}^*) = 0$ . Réciproquement, si  $\nabla f(\mathbf{x}^*) = 0$ , d'après la caractérisation de la convexité ci-dessus :

$$\forall \mathbf{y} \in E, f(\mathbf{y}) \geq f(\mathbf{x}^*) + (\mathbf{y} - \mathbf{x}^*) \cdot 0$$

et  $\mathbf{x}^*$  est minimum global. □

Nous n'avons évoqué que quelques propriétés élémentaires de l'optimisation convexe. Il s'agit d'un domaine très vaste, n'hésitez pas à consulter un livre dédié, comme le livre (gratuit) suivant :

S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2009.

<https://web.stanford.edu/~boyd/cvxbook/>



## B.2 Dualité de Wolfe

On considère le problème d'optimisation suivant, dit problème primal :

$$\begin{cases} \operatorname{argmin}_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) \\ \text{sous contraintes : } \forall i \in \{1, \dots, m\}, g_i(\mathbf{x}) \leq 0 \end{cases}$$

dans lequel  $f$  et les  $g_i$  sont convexes et différentiables.

Le Lagrangien  $L$  s'écrit :

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{i=1}^m \lambda_i g_i(\mathbf{x})$$

où  $\boldsymbol{\lambda}$  a pour composantes  $(\lambda_1, \dots, \lambda_m)$ , et  $\nabla_{\mathbf{x}} L$  désigne le vecteur des dérivées partielles de  $L$  par rapport aux composantes de  $\mathbf{x}$ .

On remarque que minimiser  $f(\mathbf{x})$  sous les contraintes  $g_i(\mathbf{x}) \leq 0$  est équivalent à résoudre le problème suivant :

$$\operatorname{argmin}_{\mathbf{x}} \max_{\boldsymbol{\lambda} \geq 0} L(\mathbf{x}, \boldsymbol{\lambda})$$

En effet, soit  $\mathbf{x}$  satisfait les contraintes et alors  $\max_{\boldsymbol{\lambda} \geq 0} L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x})$  (car pour maximiser  $L(\mathbf{x}, \boldsymbol{\lambda})$  il faut annuler les  $\lambda_i$ , les  $g_i(\mathbf{x})$  étant négatifs), soit  $\mathbf{x}$  ne satisfait pas les contraintes et alors  $\max_{\boldsymbol{\lambda} \geq 0} L(\mathbf{x}, \boldsymbol{\lambda}) = +\infty$  (car pour au moins un  $i$ ,  $g_i(\mathbf{x}) > 0$ ) mais dans ce cas  $\mathbf{x}$  ne sera pas minimum de  $f$ .

Le problème dual au sens de Lagrange du problème d'optimisation original consiste donc à résoudre :

$$\operatorname{argmax}_{\boldsymbol{\lambda} \geq 0} \min_{\mathbf{x} \in \mathbb{R}^n} L(\mathbf{x}, \boldsymbol{\lambda})$$

Comme  $f$  et les  $g_i$  sont convexes et les  $\lambda_i$  sont positifs,  $L(\mathbf{x}, \boldsymbol{\lambda})$  est convexe en la variable  $\mathbf{x}$ . Cette fonction, qui est différentiable, est donc minimum en  $\mathbf{x}$  qui annule le gradient  $\nabla_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda})$ , et cette condition suffit à définir  $\mathbf{x}$  optimal.

Le problème dual s'écrit donc :

$$\begin{cases} \operatorname{argmax}_{\mathbf{x} \in \mathbb{R}^n, \boldsymbol{\lambda} \geq 0} L(\mathbf{x}, \boldsymbol{\lambda}) \\ \text{sous contraintes : } \nabla_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}) = 0 \end{cases}$$

On parle de dual de Wolfe.

On suppose les contraintes du primal qualifiées (c'est le cas lorsqu'elles sont convexes comme ici : il suffit que l'ensemble des solutions réalisables soit non vide). Les conditions de Karush, Kuhn et Tucker (KKT) se traduisent par la proposition suivante :

**Proposition B.5**  $\mathbf{x}^*$  est solution du problème primal si et seulement s'il existe  $\boldsymbol{\lambda}^* \geq 0$  tel que :

$$\begin{cases} \nabla_{\mathbf{x}} L(\mathbf{x}^*, \boldsymbol{\lambda}^*) = 0 \\ \forall 1 \leq i \leq N, \lambda_i^* g_i(\mathbf{x}^*) = 0 \end{cases}$$

De plus,  $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$  est solution du problème dual.

La dualité de Wolfe apparaît comme un cas particulier de la dualité de Lagrange dans le cas où la fonction à minimiser et les contraintes sont convexes et différentiables.

## B.3 Optimisation numérique par algorithme de descente

### B.3.1 Exemple introductif

Considérons une fonction  $f$  dérivable et à dérivée continue, dont on cherche un minimum. La dérivée  $f'$  est positive là où  $f$  est croissante, et négative là où  $f$  est décroissante. En tout extremum, la dérivée s'annule.

Pour tout  $x_0$  du domaine de définition de  $f$ , considérons  $f(x_0 - \eta f'(x_0))$ , pour  $\eta > 0$  assez petit. Si  $f$  est croissante (respectivement décroissante) autour de  $x_0$ ,  $x_1 = x_0 - \eta f'(x_0)$  est « à gauche » (respectivement « à droite ») de  $x_0$ . Dans tous les cas,  $f(x_1) \leq f(x_0)$ .

Une autre manière de voir est d'écrire le développement limité à l'ordre 1 de  $f$  en  $x_1$  :

$$f(x_1) = f(x_0 - \eta f'(x_0)) = f(x_0) - \eta f'(x_0)^2 + o(|x_1 - x_0|)$$

On a bien  $f(x_1) \leq f(x_0)$  pour  $x_1$  au voisinage de  $x_0$  (donc pour  $\eta$  « pas trop grand »).

En définissant par récurrence :

$$\forall n \in \mathbb{N}^*, x_{n+1} = x_n - \eta f'(x_n)$$

les valeurs de  $f(x_n)$  sont décroissantes jusqu'à un certain rang  $n^*$  au delà duquel la suite  $(x_n)$  est susceptible d'osciller autour de l'optimum cherché. Il faut donc définir un critère d'arrêt pour éviter de boucler indéfiniment. Par exemple, on peut définir à l'avance un petit  $\tau > 0$  et arrêter l'algorithme lorsque  $|f'(x_n)| \leq \tau$ . Par ailleurs, le comportement de l'algorithme dépend fortement du pas  $\eta$  : un trop grand pas peut entraîner des oscillations de grande amplitude autour de l'optimum que l'on n'atteint jamais, et un trop petit pas impliquera une convergence lente.

Bien entendu, par cette méthode on ne peut trouver qu'un minimum local : selon le point de départ  $x_0$  on pourrait converger vers un autre minimum local.

La figure B.1 illustre la discussion.

### B.3.2 Algorithme de descente

Le raisonnement précédent se généralise à une fonction  $f$  définie sur un domaine de  $\mathbb{R}^d$ .

À partir de  $\mathbf{x}_0$  dans le domaine de  $f$ , on définit par récurrence :

$$\forall n \in \mathbb{N}^*, \mathbf{x}_{n+1} = \mathbf{x}_n - \eta \nabla f(\mathbf{x}_n)$$

où  $\eta > 0$  est un paramètre et  $\nabla f(\mathbf{x})$  désigne le gradient de  $f$  en  $\mathbf{x}$  : c'est le vecteur fait des  $d$  dérivées partielles de  $f$  en  $\mathbf{x}$ .

Autrement écrit, pour la  $i$ -ème composante ( $1 \leq i \leq d$ ) :

$$\forall n \in \mathbb{N}^*, x_{n+1}^i = x_n^i - \eta \frac{\partial f_i}{\partial x^i}(\mathbf{x}_n)$$

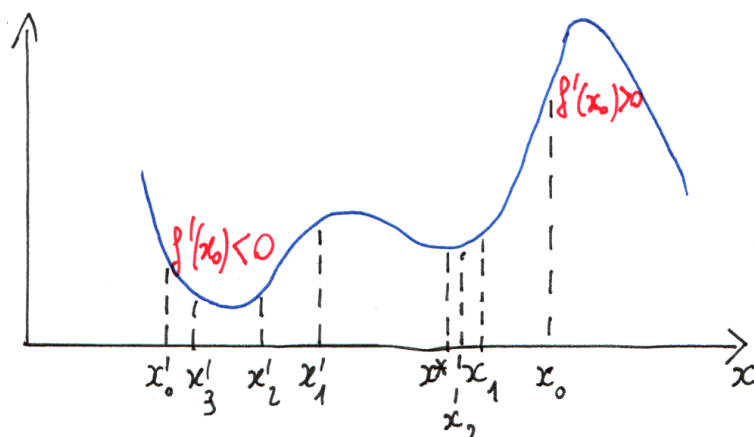


FIGURE B.1 – Illustration de la recherche du minimum d'une fonction réelle. La suite des  $(x_n)$  a l'air de converger vers le minimum local  $x^*$ . Par contre, la suite des  $(x'_n)$  oscille autour d'un minimum local à cause d'un pas  $\eta$  choisi trop grand. Rappelons que plus la pente du graphe de  $f$  est prononcée en  $x$ , plus  $|f'(x)|$  est grand et donc plus l'écart  $\eta|f'(x_n)|$  entre  $x_n$  et  $x_{n+1}$  est grand.

Si le pas  $\eta$  est assez petit, un développement limité nous fournit pour tout  $n \in \mathbb{N}$ ,  $f(\mathbf{x}_{n+1}) = f(\mathbf{x}_n - \eta \nabla f(\mathbf{x}_n)) = f(\mathbf{x}_n) - \eta \nabla f(\mathbf{x}_n) \cdot \nabla f(\mathbf{x}_n) + o(\|\mathbf{x}_{n+1} - \mathbf{x}_n\|)$ , soit, à l'ordre 1 :

$$f(\mathbf{x}_{n+1}) = f(\mathbf{x}_n) - \eta \|\nabla f(\mathbf{x}_n)\|_2^2$$

Les valeurs successives de  $f(x_n)$  sont décroissantes jusqu'à un certain rang pour  $\eta$  « pas trop grand », comme dans l'exemple introductif.

Cet algorithme cherchant un minimum avec un pas  $\eta$  constant est l'algorithme du gradient à pas constant. Consultez un cours sur ce sujet pour savoir comment choisir  $\eta_n$  à chaque itération, voire choisir une matrice  $A_n$  telle que  $\mathbf{x}_{n+1} = \mathbf{x}_n - \eta A_n \nabla f(\mathbf{x}_n)$ , de manière à accélérer la convergence.

La figure B.2 illustre l'algorithme de descente. Sur la représentation graphique de la fonction, on descend le long de la direction du gradient, d'où l'expression « algorithme de descente ».

### B.3.3 Algorithme du gradient stochastique

Lorsque la fonction à optimiser s'écrit comme une somme :

$$E(\mathbf{x}) = \sum_{i=1}^N E_i(\mathbf{x})$$

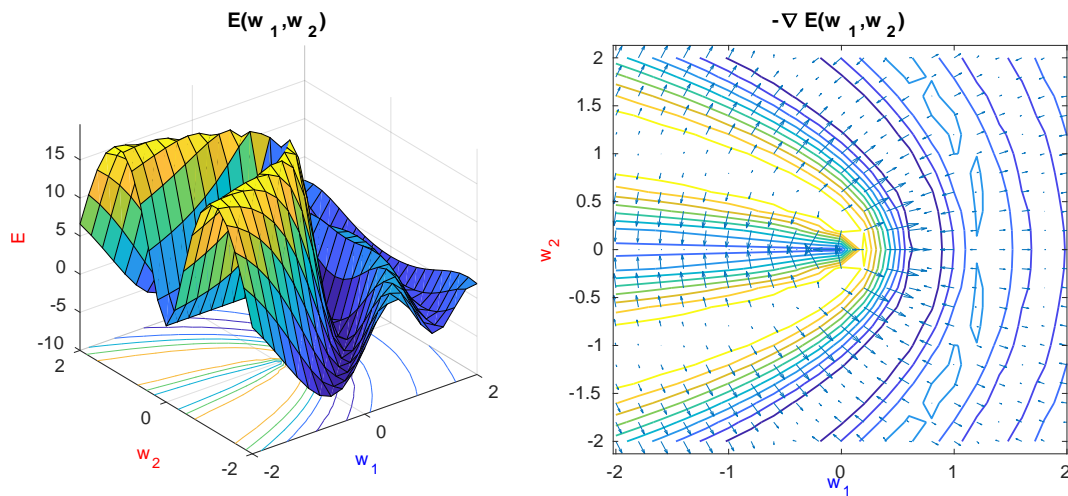


FIGURE B.2 – Illustration de la recherche du minimum d'une fonction  $E$  de deux variables  $w_1, w_2$ . La fonction est représentée sur la figure de gauche, avec des lignes de niveau dans le plan  $(w_1, w_2)$ . À droite, représentation dans le plan  $(w_1, w_2)$  des lignes de niveau et des vecteurs  $-\nabla E$ . Le gradient est orthogonal aux lignes de niveau. On voit qu'en tout point, si on se déplace dans la direction  $-\nabla E$ , alors la valeur de  $E$  diminue. En effet, sur la représentation graphique de  $E$ , on « descend ».

l'algorithme de descente s'écrit :

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \eta \sum_{i=1}^N \nabla E_i(\mathbf{x}_n)$$

par linéarité du gradient.

Il s'agit d'une situation très courante en apprentissage (les risques empiriques que l'on cherche à minimiser s'écrivent sous cette forme, en particulier pour l'entraînement des réseaux de neurones pour lesquels les poids  $\mathbf{w}$  jouent le rôle de la variable  $\mathbf{x}$ ) ainsi qu'en statistique (voir chapitre 4, section 4.2).

Lorsque  $N$  est grand, calculer  $\sum_{i=1}^N \nabla E_i$  peut poser problème :

- stocker tous les  $\nabla E_i(\mathbf{x}_n)$  avant d'en faire la somme peut prendre de la place en mémoire,
- certains termes  $\nabla E_i(\mathbf{x}_n)$  peuvent être petits devant d'autres (ou devant la somme partielle à laquelle il faut les ajouter) et ne pas être pris en compte dans la somme à cause de la précision limitée de l'arithmétique des ordinateurs<sup>1</sup>,
- il faut sommer les  $N$  gradients avant de mettre à jour  $\mathbf{x}$ . Pensez à la difficulté de procéder à l'apprentissage d'un réseau de neurones avec un nombre d'exemples  $N$  de un

1. Selon la norme IEEE 754 (implantée par tout compilateur/machine raisonnable), en double précision,  $1 + x = 1$  dès que  $x \leq 2^{-53} \approx 1.1102e - 16$ . En simple précision  $1 + x = 1$  dès que  $x \leq 2^{-24} \approx 5.96e - 8$ . Même si cela change depuis peu, les GPU ont longtemps été beaucoup plus rapides à effectuer des calculs en simple précision qu'en double précision.

million par exemple : il faudrait parcourir toute la base d'apprentissage avant de commencer à mettre à jour les poids du réseau.

Pour surmonter cette difficulté, l'algorithme stochastique de descente du gradient (ou algorithme du gradient stochastique, *stochastic gradient descent*, SGD) a été proposé.

- choisir une initialisation  $\mathbf{x}$
- Jusqu'à satisfaction d'un critère d'arrêt, répéter :
  - choisir aléatoirement une permutation  $\phi$  de  $\{1, \dots, N\}$
  - pour  $n \in \{1, \dots, N\}$ , faire :

$$\mathbf{x} \leftarrow \mathbf{x} - \eta \nabla E_{\phi(n)}(\mathbf{x})$$

Les  $E_n$  étant traités successivement, on parle d'algorithme *online*. Dans le cas d'un problème d'apprentissage, cela revient à présenter les observations étiquetées les unes après les autres après les avoir mélangées aléatoirement. À chaque étape, la valeur de la fonction  $E$  ne diminue pas forcément. Comme  $\mathbf{x}$  est mis à jour après le calcul de chaque gradient  $\nabla E_i$ ,  $\mathbf{x}$  ne se déplace pas forcément dans la direction du minimum de  $E$  cherché : on observe généralement des oscillations qui « ralentissent » la convergence. Pour limiter ce phénomène, on peut faire la mise à jour après le calcul (potentiellement en parallèle) de quelques gradients dont on moyenne l'influence. On parle alors d'algorithme par « mini-lot », compromis entre le gradient stochastique et la descente classique qui prend en compte le lot complet des  $N$  termes  $E_i$ . Il semble que ces oscillations dues au caractère stochastique de l'algorithme ont pour avantage de lui permettre de ne pas rester piégé dans des minima locaux comme l'algorithme de descente classique.

Une autre manière de réduire les oscillations dans la descente est de remplacer, dans la mise à jour  $\mathbf{x} \leftarrow \mathbf{x} - \Delta \mathbf{x}$  où

$$\Delta \mathbf{x} = \eta \nabla E_{\phi(n)}(\mathbf{x})$$

le terme  $\Delta \mathbf{x}$  en le mettant lui-même à jour par :

$$\Delta \mathbf{x} \leftarrow \beta \Delta \mathbf{x} + (1 - \beta) \eta \nabla E_{\phi(n)}(\mathbf{x})$$

où  $\beta \in [0, 1]$ . Cela permet d'amortir l'influence du gradient en le moyennant avec la dernière valeur de  $\Delta \mathbf{x}$ .

Généralement, on intègre le terme  $(1 - \beta)$  dans le pas  $\eta$  pour écrire :

$$\Delta \mathbf{x} \leftarrow \beta \Delta \mathbf{x} + \eta \nabla E_{\phi(n)}(\mathbf{x})$$

Il s'agit de la méthode du moment (*SGD with momentum*) : on fait une moyenne pondérée entre l'ancienne valeur de  $\Delta \mathbf{x}$  et le gradient courant.

La méthode du moment de Nesterov consiste à calculer :

$$\Delta \mathbf{x} \leftarrow \beta \Delta \mathbf{x} + \eta \nabla E_{\phi(n)}(\mathbf{x} - \beta \Delta \mathbf{x})$$

c'est-à-dire calculer le gradient en le point qui correspondrait à la prochaine mise à jour. Cela a pour effet d'accélérer la convergence.

De cette façon, l'algorithme du gradient stochastique accéléré par la méthode du moment de Nesterov est :

- choisir une initialisation  $\mathbf{x}$  et un pas  $\Delta\mathbf{x}$
- Jusqu'à satisfaction d'un critère d'arrêt, répéter :
  - choisir aléatoirement une permutation  $\phi$  de  $\{1, \dots, N\}$
  - pour  $n \in \{1, \dots, N\}$ , faire :

$$\begin{aligned}\Delta\mathbf{x} &\leftarrow \beta\Delta\mathbf{x} + \eta\nabla E_{\phi(n)}(\mathbf{x} - \beta\Delta\mathbf{x}) \\ \mathbf{x} &\leftarrow \mathbf{x} - \Delta\mathbf{x}\end{aligned}$$

L'algorithme du gradient stochastique a fait l'objet de nombreuses recherches, il est possible de démontrer la convergence presque sûre de cet algorithme sous certaines hypothèses raisonnables. De par son importance en apprentissage, il a donné lieu à de nombreuses variantes et travaux théoriques.

# Index

- $\epsilon$ -insensitive loss*, 118
- 0-1 loss*, 29, 60, 116
- Adaboost, 94, 115
- AIC, 72
- Algorithme
  - de descente, 128, 141, 170
  - de Lloyd, 42, 75
  - du gradient stochastique, 144, 171
  - EM, 70
- Analyse en composante principale, 28
- Apprentissage
  - automatique, 9
  - non-supervisé, 11
  - par cœur, 30, 33, 55
  - par renforcement, 10, 21
  - par transfert, 157
  - profond, 145, 149
  - supervisé, 13
- Arbre de décision, 90
- Arbre *k*-d, 83
- Astuce du noyau, 107
- Attribut, 11
- Augmentation des données, 155
- Auto-encodeur, 28, 158
- Average linkage*, 40
- Backpropagation*, 139, 141
- Bagging*, 88
- Banc de filtres, 152
- BIC, 72
- Boosting*, 94
- Capacité de généralisation, 30
- Caractéristique, 11, 19
- CART, 90
- Catégorie, 13
- Chain rule*, 162
- Chaînage, 42
- Classe, 13
- Classification
  - aux plus proches voisins, 82
  - biclasse, 14, 135
  - binaire, 14
  - hiérarchique, 39
  - multiclasse, 136
  - non-supervisée, 11, 39
  - supervisée, 13
  - un contre tous, 15
  - un contre un, 15
- Classifieur
  - de Bayes, 62
  - faible, 87
  - linéaire, 81, 102, 109, 126
  - naïf de Bayes, 28, 77
  - naïf de Bayes gaussien, 78
  - universel, 138
- Clustering*, 11, 13, 39
- Complete linkage*, 40
- Complexité de Rademacher, 38
- Concentration dans les coins, 25
- Conditions
  - de Karush, Kuhn et Tucker, 104, 170
  - de Mercer, 111
- Coniques, 163
- Convolution, 150
- Convolutional neural network*, 150
- Couche cachée, 132
- Coût
  - $\epsilon$ -insensitive*, 118
  - 0-1, 29, 36, 60, 96
  - d'erreur, 29, 56, 140
  - quadratique, 56, 116, 118
- Critère
  - de Gini, 91

- de Ward, 40
- Cross validation*, 34
- Cross-entropy loss*, 140
- Curse of dimensionality*, 23, 39, 73
- Dataset*, 10
- Decision*
  - stump*, 98
  - tree*, 90
- Deep learning*, 138, 145, 149
- Dendrogramme, 41
- Densité conditionnelle, 57
- Dérivation des fonctions composées, 162
- Dilemme
  - biais-fluctuation, 28, 89
  - biais-variance, 35
- Dimension de Vapnik-Chervonenkis, 38, 109
- Distance
  - d'édition, 40, 52
  - de Levenshtein, 40, 52
- Données
  - non-étiquetées, 10, 11
  - étiquetées, 10, 13
- Dropout*, 154
- Dualité de Wolfe, 103, 169
- Early stopping*, 145
- Échantillon, 10
- Elbow method*, 48
- Encodage un parmi  $n$ , 136
- Ensemble convexe, 167
- Entraînement, 13, 139
- Entropie croisée, 91, 140
- Epoch*, 128, 146
- Époque, 128, 146
- Équations normales, 17
- Espace de redescription, 109
- Espérance conditionnelle, 57
- Estimateur de Nadaraya-Watson, 59
- Estimation de densités, 58, 65
- ET logique, 129
- Étiquette, 10
- Évanouissement du gradient, 145
- Exponential loss*, 96, 115
- Feature space*, 109
- Feature*, 11
- Feedforward neural network*, 131
- Fenêtres de Parzen, 50, 66
- Filtre, 150
- Fine tuning*, 158
- Fonction
  - $\Gamma$ , 24
  - convexe, 18, 103, 167, 169
  - d'activation, 133
  - de perte, 29, 96, 115
  - discriminante, 15
  - logistique, 80, 134
- Forêt aléatoire, 93
- Forme normale conjonctive, 130
- Frontière de séparation, 13
- Gaussian mixture model*, 70
- Gaussian naive Bayes*, 78
- GMM, 70
- GPU, 150
- Gradient boosting*, 99
- Hard margin SVM*, 105
- Hinge loss*, 115
- Histogramme, 66
- Hiver de l'IA, 130
- Huber loss*, 63
- Hyperparamètre, 20, 34, 106, 145
- Identité de Woodbury, 122
- Imitation game*, 21
- Inertie, 42
- Ingénierie des noyaux, 111
- Intelligence artificielle, 21, 125
- Inégalité
  - de Hoeffding, 36, 161
  - de Jensen, 168
- K-means*, 42
- K-moyennes*, 42
- k-d tree*, 83
- Kernel*, 109
  - design*, 111
  - regression*, 59
  - ridge regression*, 122
  - trick*, 107
- Lagrangien, 103, 106, 107, 169
- Learning loss*, 33
- Likelihood*, 65
- Logistic loss*, 96, 115
- Loss function*, 29, 56, 96, 115
- M-estimateur*, 63



*Machine learning*, 9  
 Machine à vecteurs supports, 101  
 Malédiction de la dimension, 23, 39, 73, 83  
 Marge, 101, 127  
     souple, 105  
 Matrice  
     hessienne, 165  
     jacobienne, 161  
     orthogonale, 162  
     symétrique, 162  
*Matrix inversion lemma*, 122  
 Maximum  
     a posteriori (MAP), 62  
     de vraisemblance, 65  
*Mean shift*, 49  
 Mélange de gaussiennes, 70  
 Mesure de dissimilarité, 39  
 Méthode  
     connexioniste, 130  
     de Parzen, 50, 66  
     des plus proches voisins, 67  
     du coude, 48  
     du moment, 173  
     ensembliste, 87  
     symbolique, 130  
*MiniBatch K-means*, 54  
 Minimisation du risque, 29, 56  
 Modèle  
     discriminant, 85  
     génératif, 85  
     parcimonieux, 19, 124  
 Moindres carrés, 17  
 Moment de Nesterov, 174  
*Momentum method*, 173  
*Multilayer perceptron*, 131  
  
*No-free-lunch theorem*, 121  
 Nombre de Stirling, 46  
 Norme euclidienne, 11  
 Noyau, 109  
     invariant par translation, 113  
     linéaire, 112  
     polynomial, 113  
     RBF, 113  
  
*One-hot encoding*, 136, 139  
*One-vs-all classification*, 15  
*One-vs-one classification*, 15  
 Optimisation  
     combinatoire, 64  
     convexe, 167  
 OU logique, 129  
*Overfitting*, 145  
  
 Partitionnement, 11, 39  
 Perceptron, 109, 115, 125  
     multicouche, 131  
*Perceptron loss*, 115, 128  
 Phénomène de Hughes, 26  
 Plus proches voisins, 26, 67, 82  
*Pooling*, 153  
*Posterior probability*, 60  
*Prior probability*, 60  
 Probabilité  
     a posteriori, 60  
     a priori, 60  
  
 Quadriques, 163  
  
*Random forest*, 93  
 RANSAC, 64  
 Réduction de dimension, 28  
 Réglage fin, 158  
 Régression, 15, 58  
     Lasso, 19, 21  
     linéaire, 17, 28  
     logistique, 79, 109, 114  
     non-paramétrique, 59  
     ridge, 18, 28  
     ridge à noyau, 122  
     à noyau, 59  
     à vecteurs supports, 118  
 Regroupement hiérarchique, 39  
 Régularisation, 18, 28, 107, 145  
*Reinforcement learning*, 10, 21  
 ReLU, 133, 134, 145, 153  
 Réseau de neurones  
     artificiels, 131  
     convolutif, 150  
     récurrent, 158  
     à propagation avant, 131  
 Résidu, 17  
 Rétropropagation, 139, 141  
 Risque  
     de Bayes, 63  
     de prédiction, 30, 56  
     empirique, 29, 140  
  
*Sample*, 10

Sélection de variables, 19  
Séparateur à vaste marge, 102  
Sigmoides, 80, 133, 134, 136  
*Single linkage*, 40  
*Slack variable*, 105  
*Soft k-means*, 75  
*Soft margin SVM*, 105  
SoftMax, 137  
SoftSign, 134  
Softsign, 133  
Souche de classification, 98  
Sous-apprentissage, 31, 36  
*Sparse model*, 19, 124  
*Square loss*, 116  
*Statistical learning*, 9  
*Statistical learning theory*, 36  
*Stochastic gradient descent*, 171  
*Supervised learning*, 13  
*Support vector machine*, 101  
*Support vector regression*, 118  
Sur-apprentissage, 31, 36, 145  
SVM, 101  
    linéaire, 109, 112

Tangente hyperbolique, 133  
Taux d'apprentissage, 141, 146  
Test de Turing, 21  
*Test loss*, 33  
Théorie statistique de l'apprentissage, 36  
Théorème  
    d'approximation universelle, 137  
    de Bayes, 61  
    de Cybenko, 137  
    de Novikoff, 127  
*Training*, 13, 139  
*Training loss*, 33  
Traitement du signal, 150  
*Transfer learning*, 157  
Turing, Alan, 21

*Unsupervised learning*, 11

Validation  
    *holdout*, 33  
    *leave-one-out*, 34  
    croisée, 20, 34  
*Validation loss*, 33  
*Vanishing gradient*, 145, 153  
Variable, 11, 19  
    d'écart, 105

Volume de la boule unité, 24

XGBoost, 100  
xkcd, 1  
XOR logique, 129