Introduction au Machine Learning

Chloé-Agathe Azencott

Cet ouvrage s'adresse aux étudiantes et étudiants en fin de licence et en master d'informatique ou de maths appliquées, ainsi qu'aux élèves d'école d'ingénieurs.

L'apprentissage automatique, ou *machine Learning*, est une discipline dont les outils puissants permettent aujourd'hui à de nombreux secteurs d'activité de réaliser des progrès spectaculaires grâce à l'exploitation de grands volumes de données.

Le but de cet ouvrage est de vous fournir des bases solides sur les **concepts et les algorithmes** de ce domaine en plein essor. Il vous aidera à identifier les problèmes qui peuvent être résolus par une approche machine learning, à les formaliser, à identifier les algorithmes les mieux adaptés à chaque cas, à les mettre en oeuvre, et enfin à savoir évaluer les résultats obtenus.

Ce document est la version électronique d'un ouvrage publié aux éditions Dunod dans la collection InfoSup [1], qui contient aussi 86 exercices corrigés.

1. https://www.dunod.com/sciences-techniques/introduction-au-machine-learning-0

Préambule

Le *machine learning* (apprentissage automatique) est au cœur de la science des données et de l'intelligence artificielle. Que l'on parle de transformation numérique des entreprises, de Big Data ou de stratégie nationale ou européenne, le machine learning est devenu incontournable. Ses applications sont nombreuses et variées, allant des moteurs de recherche et de la reconnaissance de caractères à la recherche en génomique, l'analyse des réseaux sociaux, la publicité ciblée, la vision par ordinateur, la traduction automatique ou encore le trading algorithmique.

À l'intersection des statistiques et de l'informatique, le machine learning se préoccupe de la modélisation des données. Les grands principes de ce domaine ont émergé des statistiques fréquentistes ou bayésiennes, de l'intelligence artificielle ou encore du traitement du signal. Dans ce livre, nous considérons que le machine learning est la science de l'apprentissage automatique d'une fonction prédictive à partir d'un jeu d'observations de données étiquetées ou non.

Ce livre se veut une introduction aux concepts et algorithmes qui fondent le machine learning, et en propose une vision centrée sur la minimisation d'un risque empirique par rapport à une classe donnée de fonctions de prédictions.

Objectifs pédagogiques : Le but de ce livre est de vous accompagner dans votre découverte du machine learning et de vous fournir les outils nécessaires à

1. identifier les problèmes qui peuvent être résolus par des approches de machine learning ;
2. formaliser ces problèmes en termes de machine learning ;
3. identifier les algorithmes classiques les plus appropriés pour ces problèmes et les mettre en œuvre ;
4. implémenter ces algorithmes par vous-même afin d'en comprendre les tenants et aboutissants ;
5. évaluer et comparer de la manière la plus objective possible les performances de plusieurs algorithmes de machine learning pour une application particulière.

Public visé : Ce livre s'adresse à des étudiants en informatique ou maths appliquées, niveau L3 ou M1 (ou deuxième année d'école d'ingénieur), qui cherchent à comprendre les fondements des principaux algorithmes utilisés en machine learning. Il se base sur mes cours à CentraleSupélec [2] et sur OpenClassrooms [3] et suppose les prérequis suivants :

— algèbre linéaire (inversion de matrice, théorème spectral, décomposition en valeurs propres et vecteurs propres).
— notions de probabilités (variable aléatoire, distributions, théorème de Bayes).

2. http://tinyurl.com/ma2823-2017
3. https://openclassrooms.com/paths/data-scientist

Plan du livre : Ce livre commence par une vue d'ensemble du machine learning et des différents types de problèmes qu'il permet de résoudre. Il présente comment ces problèmes peuvent être formulés mathématiquement comme des problèmes d'optimisation (chapitre 1) et pose en appendice les bases d'optimisation convexe nécessaires à la compréhension des algorithmes présentés par la suite. La majeure partie de ce livre concerne les problèmes d'apprentissage supervisé ; le chapitre 2 détaille plus particulièrement leur formulation et introduit les notions d'espace des hypothèses, de risque et perte, et de généralisation. Avant d'étudier les algorithmes d'apprentissage supervisé les plus classiques et fréquemment utilisés, il est essentiel de comprendre comment évaluer un modèle sur un jeu de données, et de savoir sélectionner le meilleur modèle parmi plusieurs possibilités, ce qui est le sujet du chapitre 3.

Il est enfin pertinent à ce stade d'aborder l'entraînement de modèles prédictifs supervisés. Le livre aborde tout d'abord les modèles paramétriques, dans lesquels la fonction modélisant la distribution des données ou permettant de faire des prédictions a une forme analytique explicite. Les bases sont posées avec des éléments d'inférence bayésienne (chapitre 4), qui seront ensuite appliqués à des modèles d'apprentissage supervisé paramétriques (chapitre 5). Le chapitre 6 présente les variantes régularisées de ces algorithmes. Enfin, le chapitre 7 sur les réseaux de neurones propose de construire des modèles paramétriques beaucoup plus complexes et d'aborder les bases du deep learning.

Le livre aborde ensuite les modèles non paramétriques, à commencer par une des plus intuitives de ces approches, la méthode des plus proches voisins (chapitre 8). Suivront ensuite les approches à base d'arbres de décision, puis les méthodes à ensemble qui permettront d'introduire deux des algorithmes de machine learning supervisé les plus puissants à l'heure actuelle : les forêts aléatoires et le boosting de gradient (chapitre 9). Le chapitre 10 sur les méthodes à noyaux, introduites grâce aux machines à vecteurs de support, permettra de voir comment construire des modèles non linéaires via des modèles linéaires dans un espace de redescription des données.

Enfin, le chapitre 11 présentera la réduction de dimension, supervisée ou non supervisée, et le chapitre 12 traitera d'un des problèmes les plus importants en apprentissage non supervisé : le clustering.

Chaque chapitre sera conclu par quelques exercices.

Comment lire ce livre : Ce livre a été conçu pour être lu linéairement. Cependant, après les trois premiers chapitres, il vous sera possible de lire les suivants dans l'ordre qui vous conviendra, à l'exception du chapitre 6, qui a été écrit dans la continuité du chapitre 5. De manière générale, des références vers les sections d'autres chapitres apparaîtront si nécessaire.

Remerciements : Cet ouvrage n'aurait pas vu le jour sans Jean-Philippe Vert, qui m'a fait découvrir le machine learning, avec qui j'ai enseigné et pratiqué cette discipline pendant plusieurs années, et qui m'a fait, enfin, l'honneur d'une relecture attentive.

Ce livre doit beaucoup à ceux qui m'ont enseigné le machine learning, et plus particulièrement Pierre Baldi, Padhraic Smyth, et Max Welling ; à ceux avec qui je l'ai pratiqué, notamment les membres du Baldi Lab à UC Irvine, du MLCB et du département d'inférence empirique de l'Institut Max Planck à Tübingen, et du CBIO à Mines ParisTech, et bien d'autres encore qu'il serait difficile de tous nommer ici ; à ceux avec qui je l'ai enseigné, Karsten Borgwardt, Yannis Chaouche, Frédéric Guyon, Fabien Moutarde, mais aussi Judith Abecassis, Eugene Belilovsky, Joseph Boyd, Peter Naylor, Benoît Playe, Mihir Sahasrabudhe, Jiaqian Yu, et Luc Bertrand ; et enfin à ceux auxquels je l'ai enseigné, en particulier les étudiants du cours Data Mining

in der Bioinformatik de l'Université de Tübingen qui ont subi ma toute première tentative d'enseignement des méthodes à noyaux en 2012, et les étudiants centraliens qui ont essuyé les plâtres de la première version de ce cours à l'automne 2015.

Mes cours sont le résultat de nombreuses sources d'inspirations accumulées au cours des années. Je remercie tout particulièrement Ethem Alpaydin, David Barber, Christopher M. Bishop, Stephen Boyd, Hal Daumé III, Jerome Friedman, Trevor Hastie, Tom Mitchell, Bernhard Schölkopf, Alex Smola, Robert Tibshirani, Lieven Vandenberghe, et Alice Zhang pour leurs ouvrages.

Parce que tout serait différent sans scikit-learn, je remercie chaleureusement tous ses core-devs, et en particulier Alexandre Gramfort, Olivier Grisel, Gaël Varoquaux et Nelle Varoquaux.

Je remercie aussi Matthew Blaschko, qui m'a poussée à l'eau, et Nikos Paragios, qui l'y a encouragé.

Parce que je n'aurais pas pu écrire ce livre seule, merci à Jean-Luc Blanc des Éditions Dunod, et à tous ceux qui ont relu tout ou partie de cet ouvrage, en particulier Judith Abecassis, Luc Bertrand, Caroline Petitjean, Denis Rousselle, Erwan Scornet.

La relecture attentive de Jean-Marie Monier, ainsi que les commentaires d'Antoine Brault, ont permis d'éliminer de nombreuses coquilles et approximations de la deuxième version de ce texte.

Merci à Alix Deleporte, enfin, pour ses relectures et son soutien.

Table des matières

1 Présentation du machine learning **10**
- 1.1 Qu'est-ce que le machine learning ? . . . 10
  - 1.1.1 Pourquoi utiliser le machine learning ? . . . 11
- 1.2 Types de problèmes de machine learning . . . 13
  - 1.2.1 Apprentissage supervisé . . . 13
  - 1.2.2 Apprentissage non supervisé . . . 15
  - 1.2.3 Apprentissage semi-supervisé . . . 17
  - 1.2.4 Apprentissage par renforcement . . . 17
- 1.3 Ressources pratiques . . . 17
  - 1.3.1 Implémentations logicielles . . . 17
  - 1.3.2 Jeux de données . . . 18
- 1.4 Notations . . . 18

2 Apprentissage supervisé **20**
- 2.1 Formalisation d'un problème d'apprentissage supervisé . . . 20
  - 2.1.1 Décision . . . 21
  - 2.1.2 Classification multi-classe . . . 22
- 2.2 Espace des hypothèses . . . 23
- 2.3 Minimisation du risque empirique . . . 24
- 2.4 Fonctions de coût . . . 25
  - 2.4.1 Fonctions de coût pour la classification binaire . . . 26
  - 2.4.2 Coûts pour la classification multi-classe . . . 28
  - 2.4.3 Coûts pour la régression . . . 29
- 2.5 Généralisation et sur-apprentissage . . . 30
  - 2.5.1 Généralisation . . . 30
  - 2.5.2 Sur-apprentissage . . . 31
  - 2.5.3 Compromis biais-variance . . . 32
  - 2.5.4 Régularisation . . . 34

3 Sélection de modèle et évaluation **36**
- 3.1 Estimation empirique de l'erreur de généralisation . . . 36
  - 3.1.1 Jeu de test . . . 37
  - 3.1.2 Jeu de validation . . . 37
  - 3.1.3 Validation croisée . . . 37
  - 3.1.4 Bootstrap . . . 39
- 3.2 Critères de performance . . . 40
  - 3.2.1 Matrice de confusion et critères dérivés . . . 40
  - 3.2.2 Évaluation de méthodes de classification binaire retournant un score . . . 42

3.2.3 Erreurs de régression . . . 45
3.2.4 Comparaison à des algorithmes naïfs . . . 46

4 Inférence bayésienne 49
4.1 Modèles génératifs pour la classification binaire . . . 49
4.1.1 Inférence et prédiction . . . 50
4.1.2 Loi de Bayes . . . 50
4.1.3 Modélisation paramétrique . . . 51
4.2 Règles de décision . . . 51
4.2.1 Tests du rapport de vraisemblance . . . 52
4.2.2 Théorie de la décision bayésienne . . . 53
4.3 Estimation de densité . . . 57
4.3.1 Estimation par maximum de vraisemblance . . . 57
4.3.2 Estimateur de Bayes . . . 59
4.3.3 Décomposition biais-variance . . . 60
4.4 Classification naïve bayésienne . . . 60
4.4.1 Principes . . . 60
4.4.2 Filtrage bayésien du spam . . . 61
4.5 Sélection de modèle bayésienne . . . 62

5 Régressions paramétriques 64
5.1 Apprentissage supervisé d'un modèle paramétrique . . . 64
5.1.1 Modèles paramétriques . . . 64
5.1.2 Estimation par maximum de vraisemblance et méthode des moindres carrés . . . 66
5.2 Régression linéaire . . . 66
5.2.1 Formulation . . . 66
5.2.2 Solution . . . 66
5.2.3 Théorème de Gauss-Markov . . . 67
5.3 Régression logistique . . . 68
5.3.1 Formulation . . . 69
5.3.2 Solution . . . 70
5.4 Régression polynomiale . . . 70

6 Régularisation 72
6.1 Qu'est-ce que la régularisation ? . . . 72
6.2 La régression ridge . . . 73
6.2.1 Formulation de la régression ridge . . . 73
6.2.2 Solution . . . 74
6.2.3 Chemin de régularisation . . . 74
6.2.4 Interprétation géométrique . . . 75
6.3 Le lasso . . . 76
6.3.1 Parcimonie . . . 76
6.3.2 Formulation du lasso . . . 76
6.3.3 Solution . . . 77
6.3.4 Interprétation géométrique . . . 77
6.3.5 Chemin de régularisation . . . 77
6.4 Elastic net . . . 78

7 Réseaux de neurones artificiels 81
7.1 Le perceptron . . . 81
7.1.1 Modèle . . . 82
7.1.2 Entraînement . . . 83
7.1.3 Modélisation de fonctions booléennes . . . 85
7.2 Perceptron multi-couche . . . 86
7.2.1 Architecture . . . 86
7.2.2 Approximation universelle . . . 87
7.2.3 Modéliser XOR avec un perceptron multi-couche . . . 88
7.2.4 Entraînement par rétropropagation . . . 88
7.2.5 Et le deep learning dans tout ça ? . . . 91

8 Méthode des plus proches voisins 93
8.1 Méthode du plus proche voisin . . . 93
8.1.1 Méthode . . . 93
8.1.2 Diagramme de Voronoï . . . 94
8.2 Méthode des plus proches voisins . . . 95
8.2.1 Méthode des k plus proches voisins . . . 95
8.2.2 Apprentissage paresseux . . . 96
8.2.3 Nombre de plus proches voisins . . . 96
8.2.4 Variantes . . . 97
8.3 Distances et similarités . . . 98
8.3.1 Distances . . . 98
8.3.2 Similarités entre vecteurs réels . . . 99
8.3.3 Similarités entre ensembles . . . 100
8.3.4 Similarités entre données catégoriques . . . 101
8.4 Filtrage collaboratif . . . 102

9 Arbres et forêts 104
9.1 Arbres de décision . . . 104
9.1.1 Apprentissage hiérarchique . . . 104
9.1.2 Partition de l'espace par un arbre de décision . . . 105
9.2 Comment faire pousser un arbre . . . 106
9.2.1 CART . . . 106
9.2.2 Critères d'impureté . . . 108
9.2.3 Élaguer un arbre . . . 108
9.3 Méthodes ensemblistes : la sagesse des foules . . . 109
9.3.1 Méthodes parallèles : le bagging . . . 110
9.3.2 Méthodes séquentielles : le boosting . . . 111

10 Machines à vecteurs de support et méthodes à noyaux 116
10.1 Le cas linéairement séparable : SVM à marge rigide . . . 116
10.1.1 Marge d'un hyperplan séparateur . . . 117
10.1.2 Formulation de la SVM à marge rigide . . . 118
10.1.3 Formulation duale . . . 119
10.1.4 Interprétation géométrique . . . 121
10.2 Le cas linéairement non séparable : SVM à marge souple . . . 121
10.2.1 Formulation de la SVM à marge souple . . . 121

10.2.2 Formulation duale . . . 122
10.2.3 Interprétation géométrique . . . 124
10.3 Le cas non linéaire : SVM à noyau . . . 124
10.3.1 Espace de redescription . . . 125
10.3.2 SVM dans l'espace de redescription . . . 125
10.3.3 Astuce du noyau . . . 125
10.3.4 Noyaux . . . 126
10.4 Régression ridge à noyau . . . 128

11 Réduction de dimension **132**
11.1 Motivation . . . 132
11.1.1 Visualiser les données . . . 132
11.1.2 Réduire les coûts algorithmiques . . . 133
11.1.3 Améliorer la qualité des modèles . . . 133
11.2 Sélection de variables . . . 134
11.2.1 Méthodes de filtrage . . . 134
11.2.2 Méthodes de conteneur . . . 135
11.2.3 Méthodes embarquées . . . 137
11.3 Extraction de variables . . . 137
11.3.1 Analyse en composantes principales . . . 137
11.3.2 Factorisation de la matrice des données . . . 140
11.3.3 Auto-encodeurs . . . 142
11.3.4 Autres approches non linéaires . . . 145

12 Clustering **148**
12.1 Pourquoi partitionner ses données . . . 148
12.2 Évaluer la qualité d'un algorithme de clustering . . . 149
12.2.1 La forme des clusters . . . 149
12.2.2 La stabilité des clusters . . . 152
12.2.3 Les connaissances expert . . . 152
12.3 Clustering hiérarchique . . . 153
12.3.1 Dendrogramme . . . 153
12.3.2 Construction agglomérative ou divisive . . . 154
12.3.3 Fonctions de lien . . . 154
12.3.4 Choix du nombre de clusters . . . 155
12.3.5 Complexité algorithmique . . . 155
12.4 Méthode des k-moyennes . . . 156
12.4.1 Algorithme de Lloyd . . . 156
12.4.2 Forme des clusters . . . 156
12.4.3 Variantes . . . 157
12.5 Clustering par densité . . . 158

A Notions d'optimisation convexe **162**
A.1 Convexité . . . 162
A.1.1 Ensemble convexe . . . 162
A.1.2 Fonction convexe . . . 163
A.2 Problèmes d'optimisation convexe . . . 164
A.2.1 Formulation et vocabulaire . . . 164

A.2.2 Extrema locaux et globaux . . . 165
A.3 Optimisation convexe sans contrainte . . . 166
A.3.1 Caractérisation différentielle de la convexité . . . 166
A.3.2 Caractérisation du deuxième ordre de la convexité . . . 167
A.3.3 Algorithme du gradient . . . 168
A.3.4 Recherche linéaire par rebroussement . . . 169
A.3.5 Méthode de Newton . . . 170
A.3.6 Méthode de Newton par gradient conjugué . . . 171
A.3.7 Méthodes de quasi-Newton . . . 173
A.3.8 Algorithme du gradient stochastique . . . 174
A.3.9 Descente de coordonnées . . . 175
A.4 Optimisation convexe sous contraintes . . . 175
A.4.1 Lagrangien . . . 175
A.4.2 Dualité faible . . . 176
A.4.3 Dualité forte . . . 177
A.4.4 Conditions de Karush-Kuhn-Tucker . . . 178
A.4.5 Programmes quadratiques . . . 179

Chapitre 1

Présentation du machine learning

Le machine learning est un domaine captivant. Issu de nombreuses disciplines comme les statistiques, l'optimisation, l'algorithmique ou le traitement du signal, c'est un champ d'études en mutation constante qui s'est maintenant imposé dans notre société. Déjà utilisé depuis des décennies dans la reconnaissance automatique de caractères ou les filtres anti-spam, il sert maintenant à protéger contre la fraude bancaire, recommander des livres, films, ou autres produits adaptés à nos goûts, identifier les visages dans le viseur de notre appareil photo, ou traduire automatiquement des textes d'une langue vers une autre.

Dans les années à venir, le machine learning nous permettra vraisemblablement d'améliorer la sécurité routière (y compris grâce aux véhicules autonomes), la réponse d'urgence aux catastrophes naturelles, le développement de nouveaux médicaments, ou l'efficacité énergétique de nos bâtiments et industries.

Le but de ce chapitre est d'établir plus clairement ce qui relève ou non du machine learning, ainsi que des branches de ce domaine dont cet ouvrage traitera.

Objectifs

— Définir le machine learning

— Identifier si un problème relève ou non du machine learning

— Donner des exemples de cas concrets relevant de grandes classes de problèmes de machine learning.

1.1 Qu'est-ce que le machine learning ?

Qu'est-ce qu'apprendre, comment apprend-on, et que cela signifie-t-il pour une machine ? La question de l'*apprentissage* fascine les spécialistes de l'informatique et des mathématiques tout autant que neurologues, pédagogues, philosophes ou artistes.

Une définition qui s'applique à un programme informatique comme à un robot, un animal de compagnie ou un être humain est celle proposée par Fabien Benureau (2015) : « *L'apprentissage est une modification d'un comportement sur la base d'une expérience* ».

Dans le cas d'un programme informatique, qui est celui qui nous intéresse dans cet ouvrage, on parle d'*apprentissage automatique*, ou *machine learning*, quand ce programme a la capacité d'apprendre sans que cette modification ne soit explicitement programmée. Cette définition est celle donnée par Arthur Samuel

(1959). On peut ainsi opposer un programme *classique*, qui utilise une procédure et les données qu'il reçoit en entrée pour produire en sortie des réponses, à un programme *d'apprentissage automatique*, qui utilise les données et les réponses afin de produire la procédure qui permet d'obtenir les secondes à partir des premières.

Exemple

Supposons qu'une entreprise veuille connaître le montant total dépensé par un client ou une cliente à partir de ses factures. Il suffit d'appliquer un algorithme classique, à savoir une simple addition : un algorithme d'apprentissage n'est pas nécessaire.

Supposons maintenant que l'on veuille utiliser ces factures pour déterminer quels produits le client est le plus susceptible d'acheter dans un mois. Bien que cela soit vraisemblablement lié, nous n'avons manifestement pas toutes les informations nécessaires pour ce faire. Cependant, si nous disposons de l'historique d'achat d'un grand nombre d'individus, il devient possible d'utiliser un algorithme de machine learning pour qu'il en tire un modèle prédictif nous permettant d'apporter une réponse à notre question.

1.1.1 Pourquoi utiliser le machine learning ?

Le machine learning peut servir à résoudre des problèmes
- que l'on ne sait pas résoudre (comme dans l'exemple de la prédiction d'achats ci-dessus) ;
- que l'on sait résoudre, mais dont on ne sait formaliser en termes algorithmiques comment nous les résolvons (c'est le cas par exemple de la reconnaissance d'images ou de la compréhension du langage naturel) ;
- que l'on sait résoudre, mais avec des procédures beaucoup trop gourmandes en ressources informatiques (c'est le cas par exemple de la prédiction d'interactions entre molécules de grande taille, pour lesquelles les simulations sont très lourdes).

Le machine learning est donc utilisé quand les *données* sont abondantes (relativement), mais les *connaissances* peu accessibles ou peu développées.

Ainsi, le machine learning peut aussi aider les humains à apprendre : les modèles créés par des algorithmes d'apprentissage peuvent révéler l'importance relative de certaines informations ou la façon dont elles interagissent entre elles pour résoudre un problème particulier. Dans l'exemple de la prédiction d'achats, comprendre le modèle peut nous permettre d'analyser quelles caractéristiques des achats passés permettent de prédire ceux à venir. Cet aspect du machine learning est très utilisé dans la recherche scientifique : quels gènes sont impliqués dans le développement d'un certain type de tumeur, et comment ? Quelles régions d'une image cérébrale permettent de prédire un comportement ? Quelles caractéristiques d'une molécule en font un bon médicament pour une indication particulière ? Quels aspects d'une image de télescope permettent d'y identifier un objet astronomique particulier ?

Ingrédients du machine learning

Le machine learning repose sur deux piliers fondamentaux :
- d'une part, les *données*, qui sont les exemples à partir duquel l'algorithme va apprendre ;
- d'autre part, l'*algorithme d'apprentissage*, qui est la procédure que l'on fait tourner sur ces données pour produire un modèle. On appelle *entraînement* le fait de faire tourner un algorithme d'apprentissage sur un jeu de données.

Ces deux piliers sont aussi importants l'un que l'autre. D'une part, aucun algorithme d'apprentissage ne pourra créer un bon modèle à partir de données qui ne sont pas pertinentes – c'est le concept *garbage in, garbage out* qui stipule qu'un algorithme d'apprentissage auquel on fournit des données de mauvaise qualité

ne pourra rien en faire d'autre que des prédictions de mauvaise qualité. D'autre part, un modèle appris avec un algorithme inadapté sur des données pertinentes ne pourra pas être de bonne qualité.

Cet ouvrage est consacré au deuxième de ces piliers – les algorithmes d'apprentissage. Néanmoins, il ne faut pas négliger qu'une part importante du travail de *machine learner* ou de *data scientist* est un travail d'ingénierie consistant à préparer les données afin d'éliminer les données aberrantes, gérer les données manquantes, choisir une représentation pertinente, etc.

Attention

Bien que l'usage soit souvent d'appeler les deux du même nom, il faut distinguer l'*algorithme d'apprentissage* automatique du *modèle appris* : le premier utilise les données pour produire le second, qui peut ensuite être appliqué comme un programme classique.

Un algorithme d'apprentissage permet donc de *modéliser* un phénomène à partir d'*exemples*. Nous considérons ici qu'il faut pour ce faire définir et *optimiser* un objectif. Il peut par exemple s'agir de minimiser le nombre d'erreurs faites par le modèle sur les exemples d'apprentissage. Cet ouvrage présente en effet les algorithmes les plus classiques et les plus populaires sous cette forme.

Exemple

Voici quelques exemples de reformulation de problèmes de machine learning sous la forme d'un problème d'optimisation. La suite de cet ouvrage devrait vous éclairer sur la formalisation mathématique de ces problèmes, formulés ici très librement.

— un vendeur en ligne peut chercher à *modéliser* des types représentatifs de clientèle, à partir des transactions passées, en *maximisant* la proximité entre clients et clientes affectés à un même type ;
— une compagnie automobile peut chercher à *modéliser* la trajectoire d'un véhicule dans son environnement, à partir d'enregistrements vidéo de voitures, en *minimisant* le nombre d'accidents ;
— des chercheurs en génétique peuvent vouloir *modéliser* l'impact d'une mutation sur une maladie, à partir de données patient, en *maximisant* la cohérence de leur modèle avec les connaissances de l'état de l'art ;
— une banque peut vouloir *modéliser* les comportements à risque, à partir de son historique, en *maximisant* le taux de détection de non solvabilité.

Ainsi, le machine learning repose d'une part sur les mathématiques, et en particulier les statistiques, pour ce qui est de la construction de modèles et de leur inférence à partir de données, et d'autre part sur l'informatique, pour ce qui est de la représentation des données et de l'implémentation efficace d'algorithmes d'optimisation. De plus en plus, les quantités de données disponibles imposent de faire appel à des architectures de calcul et de base de données distribuées. C'est un point important mais que nous n'abordons pas dans cet ouvrage.

Et l'intelligence artificielle, dans tout ça ?

Le machine learning peut être vu comme une branche de l'intelligence artificielle. En effet, un système incapable d'apprendre peut difficilement être considéré comme intelligent. La capacité à apprendre et à tirer parti de ses expériences est en effet essentielle à un système conçu pour s'adapter à un environnement changeant.

L'intelligence artificielle, définie comme l'ensemble des techniques mises en œuvre afin de construire des machines capables de faire preuve d'un comportement que l'on peut qualifier d'intelligent, fait aussi appel aux sciences cognitives, à la neurobiologie, à la logique, à l'électronique, à l'ingénierie et bien plus encore.

Probablement parce que le terme « intelligence artificielle » stimule plus l'imagination, il est cependant de plus en plus souvent employé en lieu et place de celui d'apprentissage automatique.

1.2 Types de problèmes de machine learning

Le machine learning est un champ assez vaste, et nous dressons dans cette section une liste des plus grandes classes de problèmes auxquels il s'intéresse.

1.2.1 Apprentissage supervisé

L'apprentissage supervisé est peut-être le type de problèmes de machine learning le plus facile à appréhender : son but est d'apprendre à faire des *prédictions*, à partir d'une liste d'exemples *étiquetés*, c'est-à-dire accompagnés de la valeur à prédire (voir figure 1.1). Les étiquettes servent de « professeur » et supervisent l'apprentissage de l'algorithme.

Définition 1.1 (Apprentissage supervisé) On appelle *apprentissage supervisé* la branche du machine learning qui s'intéresse aux problèmes pouvant être formalisés de la façon suivante : étant données n *observations* $\{\vec{x}^{\,i}\}_{i=1,\dots,n}$ décrites dans un espace $\mathcal{X}$, et leurs *étiquettes* $\{y^i\}_{i=1,\dots,n}$ décrites dans un espace $\mathcal{Y}$, on suppose que les étiquettes peuvent être obtenues à partir des observations grâce à une fonction $\phi : \mathcal{X} \to \mathcal{Y}$ fixe et inconnue : $y^i = \phi(\vec{x}^{\,i}) + \epsilon_i$, où ϵ_i est un bruit aléatoire. Il s'agit alors d'utiliser les données pour déterminer une fonction $f : \mathcal{X} \to \mathcal{Y}$ telle que, pour tout couple $(\vec{x}, \phi(\vec{x})) \in \mathcal{X} \times \mathcal{Y}$, $f(\vec{x}) \approx \phi(\vec{x})$. ■

L'espace sur lequel sont définies les données est le plus souvent $\mathcal{X} = \mathbb{R}^p$. Nous verrons cependant aussi comment traiter d'autres types de représentations, comme des variables binaires, discrètes, catégoriques, voire des chaînes de caractères ou des graphes.

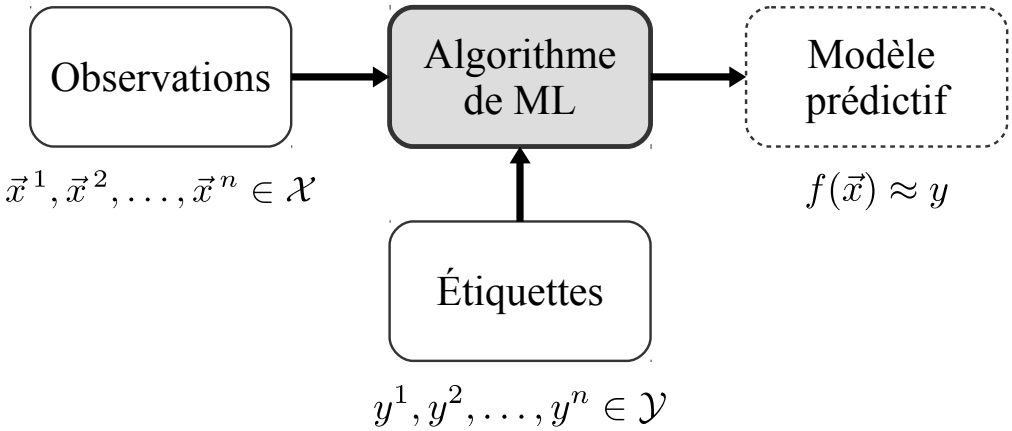


FIGURE 1.1 – Apprentissage supervisé.

Classification binaire

Dans le cas où les étiquettes sont *binaires*, elles indiquent l'appartenance à une *classe*. On parle alors de *classification binaire*.

Définition 1.2 (Classification binaire) Un problème d'apprentissage supervisé dans lequel l'espace des étiquettes est binaire, autrement dit $\mathcal{Y} = \{0, 1\}$ est appelé un problème de *classification binaire*. ■

Exemple

Voici quelques exemples de problèmes de classification binaire :
— Identifier si un email est un spam ou non ;
— Identifier si un tableau a été peint par Picasso ou non ;
— Identifier si une image contient ou non une girafe ;
— Identifier si une molécule peut ou non traiter la dépression ;
— Identifier si une transaction financière est frauduleuse ou non.

Classification multi-classe

Dans le cas où les étiquettes sont *discrètes*, et correspondent donc à plusieurs [1] *classes*, on parle de *classification multi-classe*.

Définition 1.3 (Classification multi-classe) Un problème d'apprentissage supervisé dans lequel l'espace des étiquettes est discret et fini, autrement dit $\mathcal{Y} = \{1, 2, \ldots, C\}$ est appelé un problème de *classification multi-classe*. C est le nombre de classes. ■

Exemple

Voici quelques exemples de problèmes de classification multi-classe :
— Identifier en quelle langue un texte est écrit ;
— Identifier lequel des 10 chiffres arabes est un chiffre manuscrit
— Identifier l'expression d'un visage parmi une liste prédéfinie de possibilités (colère, tristesse, joie, etc.) ;
— Identifier à quelle espèce appartient une plante ;
— Identifier les objets présents sur une photographie.

Régression

Dans le cas où les étiquettes sont à valeurs *réelles*, on parle de *régression*.

Définition 1.4 (Régression) Un problème d'apprentissage supervisé dans lequel l'espace des étiquettes est $\mathcal{Y} = \mathbb{R}$ est appelé un problème de *régression*. ■

Exemple

Voici quelques exemples de problèmes de régression :
— Prédire le nombre de clics sur un lien ;
— Prédire le nombre d'utilisateurs et utilisatrices d'un service en ligne à un moment donné ;
— Prédire le prix d'une action en bourse ;
— Prédire l'affinité de liaison entre deux molécules ;
— Prédire le rendement d'un plant de maïs.

1. Nous utilisons ici la définition bourbakiste de « plusieurs », c'est-à-dire strictement supérieur à deux.

Régression structurée

Dans le cas où l'espace des étiquettes est un espace structuré plus complexe que ceux évoqués précédemment, on parle de *régression structurée* – en anglais, *structured regression*, ou *structured output prediction*. Il peut par exemple s'agir de prédire des vecteurs, des images, des graphes, ou des séquences. La régression structurée permet de formaliser de nombreux problèmes, comme ceux de la traduction automatique ou de la reconnaissance vocale (text-to-speech et speech-to-text, par exemple). Ce cas dépasse cependant le cadre du présent ouvrage, et nous nous concentrerons sur les problèmes de classification binaire et multi-classe, ainsi que de régression classique.

L'apprentissage supervisé est le sujet principal de cet ouvrage, et sera traité du chapitre 2 au chapitre 9.

1.2.2 Apprentissage non supervisé

Dans le cadre de l'apprentissage *non supervisé*, les données ne sont pas étiquetées. Il s'agit alors de modéliser les observations pour mieux les comprendre (voir figure 1.2).

Définition 1.5 (Apprentissage non supervisé) On appelle *apprentissage non supervisé* la branche du machine learning qui s'intéresse aux problèmes pouvant être formalisés de la façon suivante : étant données n *observations* $\{\vec{x}^{\,i}\}_{i=1,\dots,n}$ décrites dans un espace $\mathcal{X}$, il s'agit d'apprendre une fonction sur $\mathcal{X}$ qui vérifie certaines propriétés. ■

FIGURE 1.2 – Apprentissage non supervisé.

Cette définition est très vague, et sera certainement plus claire sur des exemples.

Clustering

Tout d'abord, le *clustering*, ou *partitionnement*, consiste à identifier des *groupes* dans les données (voir figure 1.3). Cela permet de comprendre leurs caractéristiques générales, et éventuellement d'inférer les propriétés d'une observation en fonction du groupe auquel elle appartient.

Définition 1.6 (Partitionnement) On appelle *partitionnement* ou *clustering* un problème d'apprentissage non supervisé pouvant être formalisé comme la recherche d'une partition $\bigcup_{k=1}^{K} \mathcal{C}_k$ des n *observations* $\{\vec{x}^{\,i}\}_{i=1,\dots,n}$. Cette partition doit être pertinente au vu d'un ou plusieurs critères à préciser. ■

Exemple

Voici quelques exemples de problèmes de partitionnement

— La *segmentation de marché* consiste à identifier des groupes d'usagers ou de clients ayant un comportement similaire. Cela permet de mieux comprendre leur profil, et cibler une campagne de publicité, des contenus ou des actions spécifiquement vers certains groupes.

— Identifier des groupes de documents ayant un sujet similaire, sans les avoir au préalable étiquetés par sujet. Cela permet d'organiser de larges banques de textes.

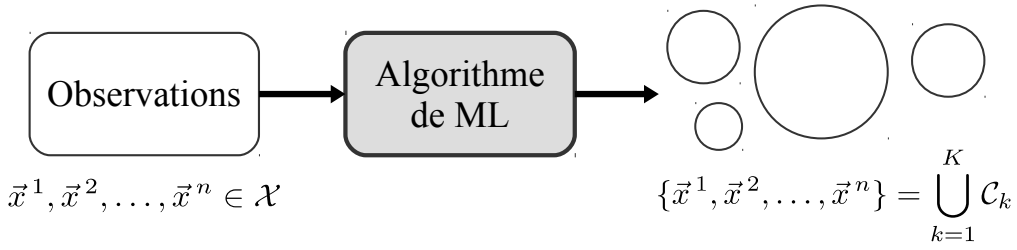


FIGURE 1.3 – Partitionnement des données, ou clustering.

— La *compression d'image* peut être formulée comme un problème de partitionnement consistant à regrouper des pixels similaires pour ensuite les représenter plus efficacement.
— La *segmentation d'image* consiste à identifier les pixels d'une image appartenant à la même région.
— Identifier des groupes parmi les patients présentant les mêmes symptômes permet d'identifier des *sous-types* d'une maladie, qui pourront être traités différemment.

Ce sujet est traité en détail au chapitre 12.

Réduction de dimension

La *réduction de dimension* est une autre famille importante de problèmes d'apprentissage non supervisé. Il s'agit de trouver une représentation des données dans un espace de dimension plus faible que celle de l'espace dans lequel elles sont représentées à l'origine (voir figure 1.4). Cela permet de réduire les temps de calcul et l'espace mémoire nécessaire au stockage les données, mais aussi souvent d'améliorer les performances d'un algorithme d'apprentissage supervisé entraîné par la suite sur ces données.

Définition 1.7 (Réduction de dimension) On appelle *réduction de dimension* un problème d'apprentissage non supervisé pouvant être formalisé comme la recherche d'un espace $\mathcal{Z}$ de dimension plus faible que l'espace $\mathcal{X}$ dans lequel sont représentées n *observations* $\{\vec{x}^{\,i}\}_{i=1,\ldots,n}$. Les projections $\{\vec{z}^{\,i}\}_{i=1,\ldots,n}$ des données sur $\mathcal{Z}$ doivent vérifier certaines propriétés à préciser. ■

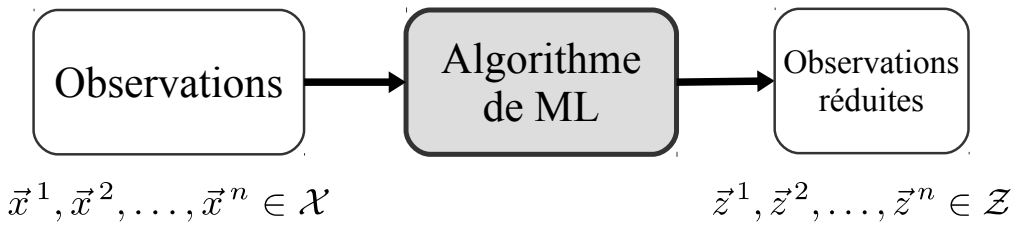


FIGURE 1.4 – Réduction de dimension.

Remarque

Certaines méthodes de réduction de dimension sont supervisées : il s'agit alors de trouver la représentation la plus pertinente *pour prédire une étiquette donnée.*

Nous traiterons de la réduction de dimension au chapitre 11.

Estimation de densité

Enfin, une grande famille de problèmes d'apprentissage non supervisé est en fait un problème traditionnel en statistiques : il s'agit d'estimer une loi de probabilité en supposant que le jeu de données en est un échantillon aléatoire. Le chapitre 4 aborde brièvement ce sujet.

1.2.3 Apprentissage semi-supervisé

Comme on peut s'en douter, l'*apprentissage semi-supervisé* consiste à apprendre des étiquettes à partir d'un jeu de données partiellement étiqueté. Le premier avantage de cette approche est qu'elle permet d'éviter d'avoir à étiqueter l'intégralité des exemples d'apprentissage, ce qui est pertinent quand il est facile d'accumuler des données mais que leur étiquetage requiert une certaine quantité de travail humain. Prenons par exemple la classification d'images : il est facile d'obtenir une banque de données contenant des centaines de milliers d'images, mais avoir pour chacune d'entre elles l'étiquette qui nous intéresse peut requérir énormément de travail. De plus, les étiquettes données par des humains sont susceptibles de reproduire des biais humains, qu'un algorithme entièrement supervisé reproduira à son tour. L'apprentissage semi-supervisé permet parfois d'éviter cet écueil. Il s'agit d'un sujet plus avancé, que nous ne considérerons pas dans cet ouvrage.

1.2.4 Apprentissage par renforcement

Dans le cadre de l'*apprentissage par renforcement*, le système d'apprentissage peut interagir avec son environnement et accomplir des actions. En retour de ces actions, il obtient une *récompense*, qui peut être positive si l'action était un bon choix, ou négative dans le cas contraire. La récompense peut parfois venir après une longue suite d'actions ; c'est le cas par exemple pour un système apprenant à jouer au go ou aux échecs. Ainsi, l'apprentissage consiste dans ce cas à définir une *politique*, c'est-à-dire une stratégie permettant d'obtenir systématiquement la meilleure récompense possible.

Les applications principales de l'apprentissage par renforcement se trouvent dans les jeux (échecs, go, etc) et la robotique. Ce sujet dépasse largement le cadre de cet ouvrage.

1.3 Ressources pratiques

1.3.1 Implémentations logicielles

De nombreux logiciels et librairies open source permettent de mettre en œuvre des algorithmes de machine learning. Nous en citons ici quelques uns :

- Les exemples de ce livre ont été écrits en Python, grâce à la très utilisée librairie scikit-learn (`http://scikit-learn.org`) dont le développement, soutenu entre autres par Inria et Télécom ParisTech, a commencé en 2007.
- De nombreux outils de machine learning sont implémentés en R, et recensés sur la page `http://cran.r-project.org/web/views/MachineLearning.html`
- Weka (*Waikato environment for knowledge analysis*, `https://www.cs.waikato.ac.nz/ml/weka/`) est une suite d'outils de machine learning écrits en Java et dont le développement a commencé en 1993.
- Shogun (`http://www.shogun-toolbox.org/`) interface de nombreux langages, et en particulier Python, Octave, R, et C#. Shogun, ainsi nommée d'après ses fondateurs Søren Sonnenburg et Gunnar Rätsch, a vu le jour en 1999.
- De nombreux outils spécialisés pour l'apprentissage profond et le calcul sur des architectures distribuées ont vu le jour ces dernières années. Parmi eux, TensorFlow (`https://www.tensorflow.org`) implémente de nombreux autres algorithmes de machine learning.

1.3.2 Jeux de données

De nombreux jeux de données sont disponibles publiquement et permettent de se faire la main ou de tester de nouveaux algorithmes de machine learning. Parmi les ressources incontournables, citons :

— Le répertoire de l'Université de Californie à Irvine, UCI Repository (`http://www.ics.uci.edu/~mlearn/MLRepository.html`)
— Les ressources listées sur KDNuggets : `http://www.kdnuggets.com/datasets/index.html`
— La plateforme de compétitions en sciences des données Kaggle (`https://www.kaggle.com/`)

1.4 Notations

Autant que faire se peut, nous utilisons dans cet ouvrage les notations suivantes :
— Les lettres minuscules (x) représentent un scalaire ;
— les lettres minuscules surmontées d'une flèche ($\vec{x}$) représentent un vecteur ;
— les lettres majuscules (X) représentent une matrice, un événement ou une variable aléatoire ;
— les lettres calligraphiées ($\mathcal{X}$) représentent un ensemble ou un espace ;
— les *indices* correspondent à une variable tandis que les *exposants* correspondent à une observation : x_j^i est la j-ème variable de la i-ème observation, et correspond à l'entrée X_{ij} de la matrice X ;
— n est un nombre d'observations, p un nombre de variables, C un nombre de classes ;
— $[a]_+$ représente la partie positive de $a \in \mathbb{R}$, autrement dit $\max(0, a)$.
— $\mathbb{P}(A)$ représente la probabilité de l'événement A ;
— δ est la fonction Dirac, c'est-à-dire.

$$\begin{aligned} \delta : \mathcal{X} \times \mathcal{X} &\to \{0, 1\} \\ (u, v) &\mapsto \begin{cases} 1 & \text{si } u = v \\ 0 & \text{sinon ;} \end{cases} \end{aligned}$$

— δ_z est la fonction indicatrice

$$\begin{aligned} \delta_z : \{\text{Vrai}, \text{Faux}\} &\to \{0, 1\} \\ b &\mapsto \begin{cases} 1 & \text{si } b\text{est vrai} \\ 0 & \text{sinon ;} \end{cases} \end{aligned}$$

— $\langle ., . \rangle$ représente le produit scalaire sur $\mathbb{R}^p$;
— $\langle ., . \rangle_{\mathcal{H}}$ représente le produit scalaire sur $\mathcal{H}$;
— $M \succeq 0$ signifie que M est une matrice symétrique semi-définie positive.

Points clefs

— Un algorithme de machine learning est un algorithme qui apprend un modèle à partir d'exemples, par le biais d'un problème d'optimisation.
— On utilise le machine learning lorsqu'il est difficile ou impossible de définir les instructions explicites à donner à un ordinateur pour résoudre un problème, mais que l'on dispose de nombreux exemples illustratifs.
— Les algorithmes de machine learning peuvent être divisés selon la nature du problème qu'ils cherchent à résoudre, en apprentissage supervisé, non supervisé, semi-supervisé, et par renforcement.

Pour aller plus loin

- Pour plus de détails sur l'estimation de densité, on consultera le livre de Scott (1992).
- Sur la régression structurée, on pourra se référer à BakIr et al. (2007).
- L'ouvrage de Barto (1998) est un bon point de départ pour se plonger dans le sujet de l'apprentissage par renforcement.

Bibliographie

BakIr, G., Hofmann, T., Schölkopf, B., Smola, A. J., Taskar, B., et Vishwanathan, S. V. N. (2007). *Predicting Structured Data*. MIT Press, Cambridge, MA. `https://mitpress.mit.edu/books/predicting-structured-data`.

Barto, R. S. et Sutton A. G. (1998). *Reinforcement Learning : An Introduction*. MIT Press, Cambridge, MA. `http://incompleteideas.net/book/the-book-2nd.html`.

Benureau, F. (2015). *Self-Exploration of Sensorimotor Spaces in Robots*. Thèse de doctorat, Université de Bordeaux.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 44(1.2) :206–226.

Scott, D. W. (1992). *Multivariate density estimation*. Wiley, New York.

Chapitre 2

Apprentissage supervisé

Dans cet ouvrage, nous nous intéresserons principalement aux problèmes d'apprentissage *supervisé* : il s'agit de développer des algorithmes qui soient capables d'apprendre des modèles *prédictifs*. À partir d'exemples étiquetés, ces modèles seront capables de prédire l'étiquette de nouveaux objets. Le but de ce chapitre est de développer les concepts généraux qui nous permettent de formaliser ce type de problèmes.

Objectifs

- Formaliser un problème comme un problème d'apprentissage supervisé ;
- Choisir une fonction de coût ;
- Lier la capacité d'un modèle à généraliser avec sa complexité.

2.1 Formalisation d'un problème d'apprentissage supervisé

Un problème d'*apprentissage supervisé* peut être formalisé de la façon suivante : étant données n *observations* $\{\vec{x}^{\,1}, \vec{x}^{\,2}, \ldots, \vec{x}^{\,n}\}$, où chaque observation $\vec{x}^{\,i}$ est un élément de l'espace des observations $\mathcal{X}$, et leurs *étiquettes* $\{y^1, y^2, \ldots, y^n\}$, où chaque étiquette y^i appartient à l'espace des étiquettes $\mathcal{Y}$, le but de l'apprentissage supervisé est de trouver une fonction $f : \mathcal{X} \to \mathcal{Y}$ telle que $f(\vec{x}) \approx y$, pour toutes les paires $(\vec{x}, y) \in \mathcal{X} \times \mathcal{Y}$ ayant la même relation que les paires observées. L'ensemble de $\mathcal{D} = \{(\vec{x}^{\,i}, y^i)\}_{i=1,\ldots,n}$ forme le *jeu d'apprentissage*.

Nous allons considérer dans cet ouvrage trois cas particuliers pour $\mathcal{Y}$:

- $\mathcal{Y} = \mathbb{R}$: on parle d'un problème de *régression* ;
- $\mathcal{Y} = \{0, 1\}$: on parle d'un problème de *classification binaire*, et les observations dont l'étiquette vaut 0 sont appelées *négatives* tandis que celles dont l'étiquette vaut 1 sont appelées *positives*. Dans certains cas, il sera mathématiquement plus simple d'utiliser $\mathcal{Y} = \{-1, 1\}$;
- $\mathcal{Y} = \{1, 2, \ldots, C\},\ C > 2$: on parle d'un problème de *classification multi-classe*.

Dans de nombreuses situations, on se ramènera au cas où $\mathcal{X} = \mathbb{R}^p$. On dira alors que les observations sont représentées par p *variables*. Dans ce cas, la matrice $X \in \mathbb{R}^{n \times p}$ telle que $X_{ij} = x_j^i$ soit la j-ème variable de la i-ème observation est appelée *matrice de données* ou *matrice de design*.

Le machine learning étant issu de plusieurs disciplines et champs d'applications, on trouvera plusieurs noms pour les mêmes objets. Ainsi les variables sont aussi appelées *descripteurs*, *attributs*, *prédicteurs*, ou *caractéristiques* (en anglais, *variables, descriptors, attributes, predictors* ou encore *features*). Les *observations* sont aussi appelées *exemples*, *échantillons* ou *points du jeu de donnée* (en anglais, *samples* ou *data points*). Enfin, les étiquettes sont aussi appelées *variables cibles* (en anglais, *labels, targets* ou *outcomes*).

Ces concepts sont illustrés sur la figure 2.1.

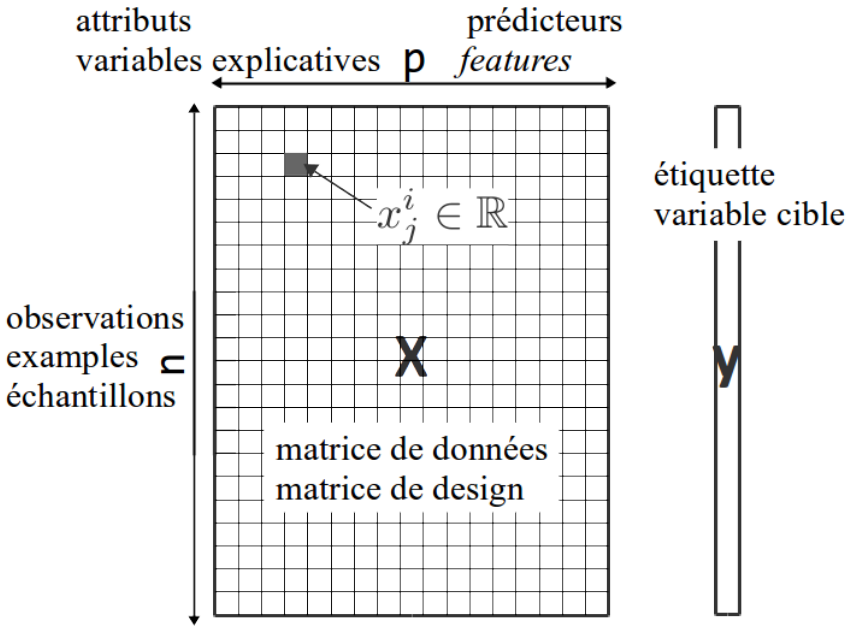


FIGURE 2.1 – Les données d'un problème d'apprentissage supervisé sont organisées en une matrice de design et un vecteur d'étiquettes. Les observations sont représentées par leurs variables explicatives.

2.1.1 Décision

Dans le cas d'un problème de classification, le modèle prédictif peut prendre directement la forme d'une fonction f à valeurs dans $\{0, 1\}$, ou utiliser une fonction intermédiaire g à valeurs réelles, qui associe à une observation un score d'autant plus élevé qu'elle est susceptible d'être positive. Ce score peut par exemple être la probabilité que cette observation appartienne à la classe positive. On obtient alors f en *seuillant* g ; g est appelée *fonction de décision*.

Définition 2.1 (Fonction de décision) Dans le cadre d'un problème de classification binaire, on appelle *fonction de décision*, ou *fonction discriminante*, une fonction $g : \mathcal{X} \mapsto \mathbb{R}$ telle que $f(\vec{x}) = 0$ si et seulement si $g(\vec{x}) \leq 0$ et $f(\vec{x}) = 1$ si et seulement si $g(\vec{x}) > 0$.

Cette définition se généralise dans le cas de la classification *multi-classe* : on a alors C fonctions de décision $g_c : \mathcal{X} \mapsto \mathbb{R}$ telles que $f(\vec{x}) = \arg\max_{c=1,\ldots,C} g_c(\vec{x})$. ■

Le concept de fonction de décision permet de partitionner l'espace en *régions de décision* :

Définition 2.2 (Région de décision) Dans le cas d'un problème de classification binaire, la fonction discriminante partitionne l'espace des observations $\mathcal{X}$ en deux *régions de décision*, $\mathcal{R}_0$ et $\mathcal{R}_1$, telles que

$$\mathcal{R}_0 = \{\vec{x} \in \mathcal{X} | g(\vec{x}) \leq 0\} \text{ et } \mathcal{R}_1 = \{\vec{x} \in \mathcal{X} | g(\vec{x}) > 0\}.$$

Dans le cas multi-classe, on a alors C régions de décision

$$\mathcal{R}_c = \{\vec{x} \in \mathcal{X} | g_c(\vec{x}) = \max_k g_k(\vec{x})\}.$$

■

Les régions de décision sont séparées par des *frontières de décision* :

Définition 2.3 (Frontière de décision) Dans le cadre d'un problème de classification, on appelle *frontière de décision*, ou *discriminant*, l'ensemble des points de $\mathcal{X}$ où une fonction de décision s'annule. Dans le cas d'un problème binaire, il y a une seule frontière de décision ; dans le cas d'un problème multi-classe à C classes, il y en a C. ■

2.1.2 Classification multi-classe

Parmi les méthodes d'apprentissage supervisé que présente cet ouvrage, certaines permettent de résoudre directement des problèmes de classification multi-classe. Cependant, tout algorithme de classification binaire peut être utilisé pour résoudre un problème de classification à C classes, par une approche *une-contre-toutes* ou une approche *une-contre-une*.

Définition 2.4 (Une-contre-toutes) Étant donné un problème de classification multi-classe à C classes, on appelle *une-contre-toutes*, ou *one-versus-all*, l'approche qui consiste à entraîner C classifieurs binaires. Le c-ème de ces classifieurs utilise tous les exemples de la classe c comme exemples positifs, et toutes les autres observations comme exemples négatifs, pour apprendre une fonction de décision g_c. Ainsi, chaque classifieur apprend à distinguer une classe de toutes les autres. L'étiquette de $\vec{x}$ est donnée par celle des fonctions de décision qui retourne le score le plus élevé :

$$f(\vec{x}) = \arg\max_{c=1,\ldots,C} g_c(\vec{x}).$$

■

Définition 2.5 (Une-contre-une) Étant donné un problème de classification multi-classe à C classes, on appelle *une-contre-une*, ou *one-versus-one*, l'approche qui consiste à créer $C(C-1)$ classifieurs binaires séparant chacun une classe d'une autre, en ignorant tous les autres exemples. Soit g_{ck} la fonction de décision du classifieur binaire qui sépare la classe c de la classe k. L'étiquette de $\vec{x}$ est déterminée selon :

$$f(\vec{x}) = \arg\max_{c=1,\ldots,C} \left(\sum_{k \neq c} g_{ck}(\vec{x}) \right).$$

■

Il est difficile de dire laquelle de ces deux approches est la plus performante. En pratique, le choix sera souvent guidé par des considérations de complexité algorithmique : est-il plus efficace d'entraîner C modèles sur n observations, ou $C(C-1)$ modèles sur $\frac{n}{C}$ observations (en supposant les classes équilibrées, autrement dit que $\mathcal{D}$ contient sensiblement autant d'exemples de chaque classe) ? Par ailleurs, on prendra aussi en compte le nombre d'exemples disponibles pour chaque classe : s'il est plus efficace d'entraîner un modèle sur peu de données, si ce nombre est trop faible, il sera difficile d'obtenir un modèle de bonne qualité.

2.2 Espace des hypothèses

Pour poser un problème d'apprentissage supervisé, il nous faut décider du type de fonctions de modélisation que nous allons considérer.

Définition 2.6 (Espace des hypothèses) On appelle *espace des hypothèses* l'espace de fonctions $\mathcal{F} \subseteq \mathcal{Y}^{\mathcal{X}}$ décrivant les fonctions de modélisation que nous allons considérer. Cet espace est choisi en fonction de nos *convictions* par rapport au problème. ■

Dans l'exemple de la figure 2.2, on pourra décider de se restreindre à des discriminants qui soient des ellipses à axes parallèles aux axes de coordonnées. Ainsi, l'espace des hypothèse sera

$$\mathcal{F} = \{\vec{x} \mapsto \alpha(x_1 - a)^2 + \beta(x_2 - b)^2 - 1\}.$$

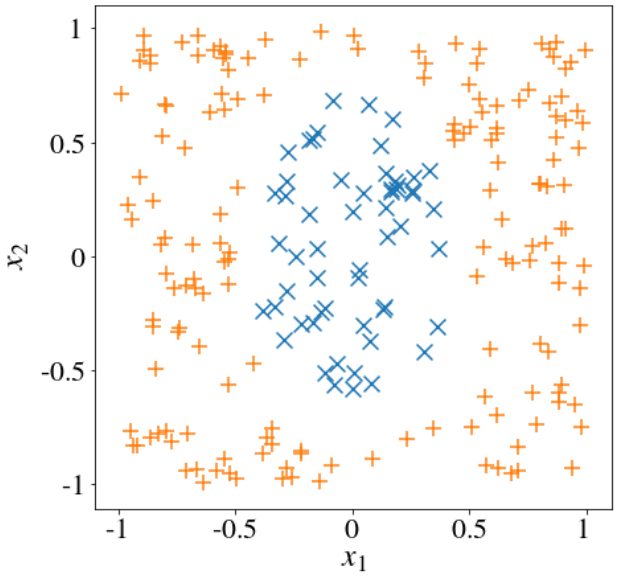


FIGURE 2.2 – Les exemples positifs (+) et négatifs (x) semblent être séparables par une ellipse.

Étant donnés un jeu de n observations étiquetées $\mathcal{D} = \{(\vec{x}^{\,i}, y^i)\}_{i=1,\dots,n}$ et un espace d'hypothèses $\mathcal{F}$. La tâche d'apprentissage supervisé consiste à supposer que les étiquettes y^i ont été calculées grâce à une fonction $\phi : \mathcal{X} \to \mathcal{Y}$, et à trouver une hypothèse $f \in \mathcal{F}$ qui approche au mieux la fonction cible ϕ. Pour réaliser une telle tâche, nous allons avoir alors besoin de deux outils supplémentaires :

1. Une façon de *quantifier la qualité d'une hypothèse*, afin de pouvoir déterminer si une hypothèse satisfaisante (voire optimale) a été trouvée. Pour cela, nous allons définir dans la section 2.4 la notion de *fonction de coût*.
2. Une façon de *chercher une hypothèse optimale* dans $\mathcal{F}$. Dans cet ouvrage, nous allons nous concentrer sur les méthodes d'*apprentissage par optimisation* : les algorithmes d'apprentissage supervisé que nous allons étudier auront pour but de trouver dans $\mathcal{F}$ l'hypothèse optimale au sens de la fonction de coût (cf. section 2.3). Différents algorithmes définiront différents $\mathcal{F}$, et selon les cas cette recherche sera exacte ou approchée.

Le choix de l'espace des hypothèses est fondamental. En effet, si cet espace ne contient pas la « bonne » fonction, par exemple si l'on choisit comme espace des hypothèses pour les données de la figure 2.2 l'ensemble des droites, il sera impossible de trouver une bonne fonction de décision. Cependant, si l'espace est trop

générique, il sera plus difficile et intensif en temps de calcul d'y trouver une bonne fonction de modélisation.

2.3 Minimisation du risque empirique

Résoudre un problème d'apprentissage supervisé revient à trouver une fonction $f \in \mathcal{F}$ dont les prédictions soient les plus proches possibles des véritables étiquettes, sur tout l'espace $\mathcal{X}$. On utilise pour formaliser cela la notion de *fonction de coût* :

Définition 2.7 (Fonction de coût) Une *fonction de coût* $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, aussi appelée *fonction de perte* ou *fonction d'erreur* (en anglais : *cost function* ou *loss function*) est une fonction utilisée pour quantifier la qualité d'une prédiction : $L(y, f(\vec{x}))$ est d'autant plus grande que l'étiquette $f(\vec{x})$ est éloignée de la vraie valeur y. ■

Étant donnée une fonction de coût L, nous cherchons donc f qui minimise ce coût sur l'ensemble des valeurs possibles de $\vec{x} \in \mathcal{X}$, ce qui est formalisé par la notion de *risque.*

Définition 2.8 (Risque) Dans le cadre d'un problème d'apprentissage supervisé, on appelle *risque* l'espérance d'une fonction de coût (voir chapitre 4 pour la modélisation probabiliste d'un problème d'apprentissage supervisé) :

$$\mathcal{R}(h) = \mathbb{E}_{\mathcal{X}}[L(h(\vec{x}), y)].$$

■

La fonction f que nous cherchons vérifie donc $f = \arg\min_{h \in \mathcal{F}} \mathbb{E}[L(h(\vec{x}), y)]$. Ce problème est généralement insoluble sans plus d'hypothèses : si nous connaissions les étiquettes de tous les points de $\mathcal{X}$, nous n'aurions pas besoin d'apprentissage automatique. Étant données n observations étiquetées $\{(\vec{x}^{\,i}, y^i)\}_{i=1,\dots,n}$, on approchera donc le risque par son estimation sur ces données observées

Définition 2.9 (Risque empirique) Dans le cadre d'un problème d'apprentissage supervisé, étant données n observations étiquetées $\{(\vec{x}^{\,i}, y^i)\}_{i=1,\dots,n}$, on appelle *risque empirique* l'estimateur

$$R_n(h) = \frac{1}{n} \sum_{i=1}^{n} L(h(\vec{x}^{\,i}), y^i).$$

■

Le prédicteur par *minimisation du risque empirique* est donc

$$f = \arg\min_{h \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} L(h(\vec{x}^{\,i}), y^i). \tag{2.1}$$

Selon le choix de $\mathcal{F}$, l'équation 2.1 peut avoir une solution analytique explicite. Cela ne sera pas souvent le cas ; cependant on choisira souvent une fonction de coût *convexe* afin de résoudre plus facilement ce problème d'optimisation (voir l'annexe A sur l'optimisation convexe).

La minimisation du risque empirique est généralement un problème *mal posé* au sens de Hadamard, c'est-à-dire qu'il n'admet pas une solution unique dépendant de façon continue des conditions initiales. Il se peut par exemple qu'un nombre infini de solutions minimise le risque empirique à zéro (voir figure 2.3).

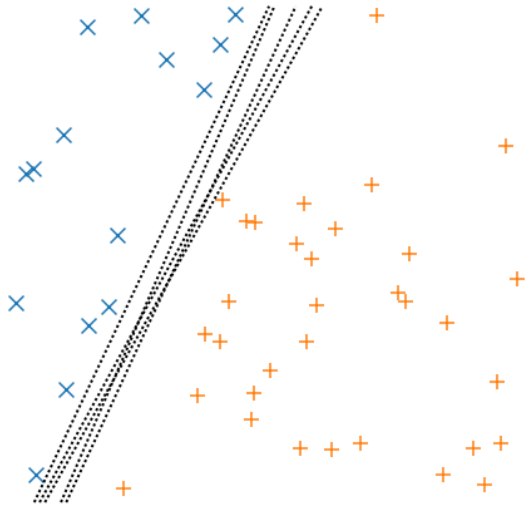


FIGURE 2.3 – Une infinité de droites séparent parfaitement les points positifs (+) des points négatifs (x). Chacune d'entre elles a un risque empirique nul.

De plus, le prédicteur par minimisation du risque empirique n'est pas statistiquement consistant. Rappelons qu'un estimateur θ_n (dépendant de n observations) d'un paramètre θ est *consistant* s'il converge en probabilité vers θ quand n croit vers l'infini :

$$\forall \epsilon > 0, \lim_{n \to \infty} \mathbb{P}(|\theta_n - \theta| \geq \epsilon) = 0. \tag{2.2}$$

La loi des grands nombres nous garantit que le risque empirique converge vers le risque :

$$\forall h \in \mathcal{F}, R_n(h) \xrightarrow[n \to \infty]{} \mathcal{R}(h). \tag{2.3}$$

Cela ne suffit cependant pas à garantir que le minimum du risque empirique $\min_{h \in \mathcal{F}} R_n(h)$ converge vers le minimum du risque. En effet, si $\mathcal{F}$ est l'espace des fonctions mesurables, $\min_{h \in \mathcal{F}} R_n(h)$ vaut généralement 0, ce qui n'est pas le cas de $\mathcal{R}(h)$. Il n'y a donc aucune garantie que la fonction f_n qui minimise $R_n(h)$ soit un bon estimateur du minimiseur f de $\mathcal{R}(h)$.

La consistance de la minimisation du risque empirique dépend de l'espace des versions $\mathcal{F}$. L'étude de cette consistance est un des principaux éléments de la théorie de l'apprentissage de Vapnik-Chervonenkis, qui dépasse l'ambition de cet ouvrage.

2.4 Fonctions de coût

Il existe de nombreuses fonctions de coût. Le choix d'une fonction de coût dépend d'une part du problème en lui-même, autrement dit de ce que l'on trouve pertinent pour le cas pratique considéré, et d'autre part de considérations pratiques : peut-on ensuite résoudre le problème d'optimisation qui résulte de ce choix de façon suffisamment exacte et rapide ? Cette section présente les fonctions de coûts les plus couramment utilisées et on pourra s'y référer tout au long de la lecture de cet ouvrage.

2.4.1 Fonctions de coût pour la classification binaire

Pour définir des fonctions de coût pour la classification binaire, on considèrera souvent $\mathcal{Y} = \{-1, 1\}$. En effet, dans le cas d'une classification parfaite, le produit $yf(\vec{x})$ est alors égal à 1.

Coût 0/1 pour la classification binaire

Définition 2.10 (Coût 0/1 – classification binaire) Dans le cas d'une fonction f à valeurs binaires, on appelle *fonction de coût 0/1*, ou *0/1 loss*, la fonction suivante :

$$\begin{aligned} L_{0/1} : \mathcal{Y} \times \mathcal{Y} &\to \mathbb{R} \\ y, f(\vec{x}) &\mapsto \begin{cases} 1 & \text{si } f(\vec{x}) \neq y \\ 0 & \text{sinon.} \end{cases} \end{aligned}$$

■

En utilisant $\mathcal{Y} = \{-1, 1\}$, on peut la réécrire de la manière suivante :

$$L_{0/1}(y, f(\vec{x})) = \frac{1 - yf(\vec{x})}{2}.$$

Quand on utilise cette fonction de coût, le risque empirique est le nombre moyen d'erreurs de prédiction.

Si l'on considère pour f une fonction de décision (à valeurs réelles) plutôt qu'une fonction de prédiction à valeurs binaires, on peut définir la fonction de coût 0/1 comme suit :

Coût 0/1 pour la régression

Définition 2.11 (Coût 0/1 – régression) Quand on considère une fonction de décision à valeurs réelles, on appelle *fonction de coût 0/1*, ou *0/1 loss*, la fonction suivante :

$$\begin{aligned} L_{0/1} : \mathcal{Y} \times \mathbb{R} &\to \mathbb{R} \\ y, f(\vec{x}) &\mapsto \begin{cases} 1 & \text{si } yf(\vec{x}) \leq 0 \\ 0 & \text{sinon.} \end{cases} \end{aligned}$$

■

L'inconvénient de cette fonction de coût est qu'elle n'est pas dérivable, ce qui compliquera les problèmes d'optimisation l'utilisant. De plus, elle n'est pas très fine : l'erreur est la même que $f(\vec{x})$ soit très proche ou très loin du seuil de décision. Rappelons que pour une classification parfaite, quand $\mathcal{Y} = \{-1, 1\}$, $yf(\vec{x}) = 1$. On peut ainsi définir une fonction de coût qui soit d'autant plus grande que $yf(\vec{x})$ s'éloigne de 1 à gauche ; on considère qu'il n'y a pas d'erreur si $yf(\vec{x}) > 1$. Cela conduit à la définition d'erreur *hinge*, ainsi appelée car elle forme un coude, ou une charnière (cf. figure 2.4).

Erreur hinge

Définition 2.12 (Erreur hinge) On appelle *fonction d'erreur hinge*, ou *hinge loss*, la fonction suivante :

$$\begin{aligned} L_{\text{hinge}} : \{-1,1\} \times \mathbb{R} &\to \mathbb{R} \\ y, f(\vec{x}) &\mapsto \begin{cases} 0 & \text{si } yf(\vec{x}) \geq 1 \\ 1 - yf(\vec{x}) & \text{sinon.} \end{cases} \end{aligned}$$

De manière plus compacte, l'erreur hinge peut aussi s'écrire

$$L_{\text{hinge}}(f(\vec{x}), y) = \max\left(0, 1 - yf(\vec{x})\right) = \left[1 - yf(\vec{x})\right]_+ .$$

■

On peut aussi considérer que $f(\vec{x})$ doit être la plus proche possible de 1 pour les observations positives (et -1 pour les observations négatives). Ainsi, on pénalisera aussi les cas où $yf(\vec{x})$ s'éloigne de 1 par la droite, ce que l'on peut faire avec le *coût quadratique*.

Coût quadratique pour la classification binaire

Définition 2.13 (Coût quadratique) Dans le cadre d'un problème de classification binaire, on appelle *coût quadratique*, ou *square loss*, la fonction suivante :

$$\begin{aligned} L_{\text{square}} : \{-1,1\} \times \mathbb{R} &\to \mathbb{R} \\ y, f(\vec{x}) &\mapsto \left(1 - yf(\vec{x})\right)^2 . \end{aligned}$$

■

Enfin, on peut chercher à définir une fonction de décision dont la valeur absolue quantifie notre confiance en sa prédiction. On cherche alors à ce que $yf(\vec{x})$ soit la plus grande possible, et on utilise le *coût logistique*.

Coût logistique

Définition 2.14 (Coût logistique) On appelle *fonction de coût logistique*, ou *logistic loss*, la fonction suivante :

$$\begin{aligned} L_{\log} : \{-1,1\} \times \mathbb{R} &\to \mathbb{R} \\ y, f(\vec{x}) &\mapsto \log\left(1 + \exp(-yf(\vec{x}))\right) . \end{aligned}$$

■

Si l'on préfère utiliser $\mathcal{Y} = \{0, 1\}$, le coût logistique est équivalent à l'*entropie croisée*.

Entropie croisée

Définition 2.15 (Entropie croisée) Dans le cas binaire, on appelle *entropie croisée*, ou *cross-entropy*, la fonction suivante :

$$\begin{aligned} L_H : \{0,1\} \times]0,1[&\to \mathbb{R} \\ y, f(\vec{x}) &\mapsto -y \log f(\vec{x}) - (1-y)\log(1 - f(\vec{x})). \end{aligned}$$

■

Remarque

L'entropie croisée est issue de la théorie de l'information, d'où son nom. En considérant que la véritable classe de $\vec{x}$ est modélisée par une distribution Q, et sa classe prédite par une distribution P, nous allons chercher à modéliser P de sorte qu'elle soit la plus proche possible de Q. On utilise pour cela la *divergence de Kullback-Leibler* :

$$\begin{aligned} KL(Q||P) &= \sum_{c=0,1} Q(y=c|\vec{x}) \log \frac{Q(y=c|\vec{x}}{P(y=c|\vec{x})} \\ &= -\sum_{c=0,1} Q(y=c|\vec{x}) \log P(y=c|\vec{x}) + \sum_{c=0,1} Q(y=c|\vec{x}) \log Q(y=c|\vec{x}) \end{aligned}$$

Comme $Q(y=c|\vec{x})$ vaut soit 0 (c n'est pas la classe de $\vec{x}$) soit 1 (dans le cas contraire), le deuxième terme de cette expression est nul et on retrouve ainsi la définition ci-dessus de l'entropie croisée.

Les fonctions de perte pour la classification binaire sont illustrées sur la figure 2.4.

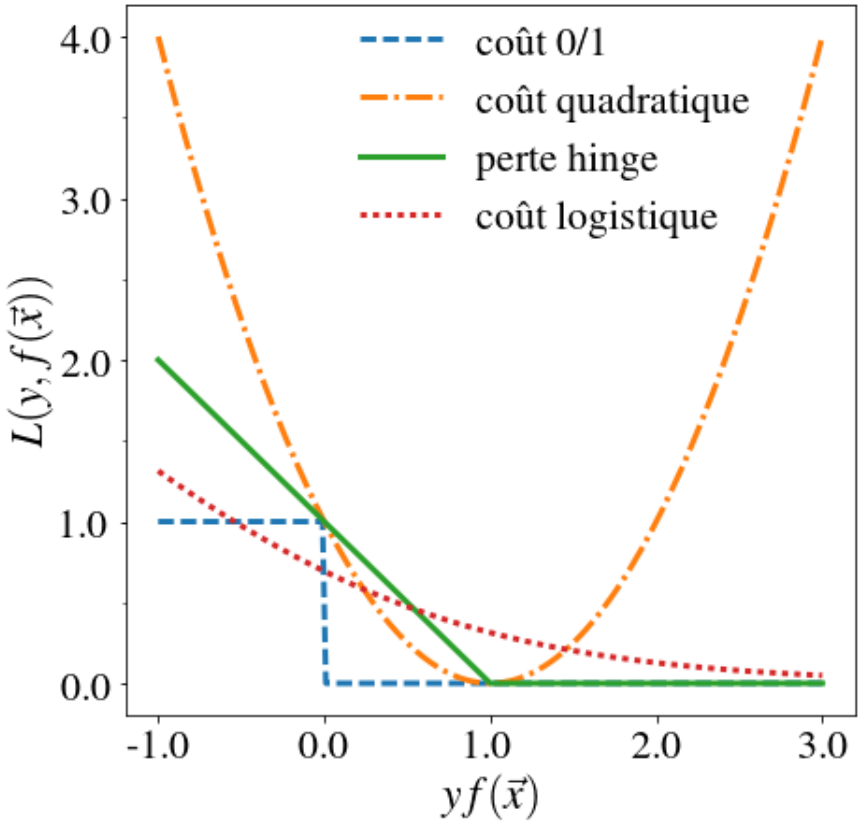


FIGURE 2.4 – Fonctions de perte pour la classification binaire.

2.4.2 Coûts pour la classification multi-classe

Entropie croisée

La définition de l'entropie croisée dans le cas binaire se généralise naturellement au cas multi-classe en considérant C fonctions de décision $f_c : \mathcal{X} \to \mathcal{Y}$.

Définition 2.16 (Entropie croisée) Dans le cas multi-classe, on appelle *entropie croisée*, ou *cross-entropy*, la fonction suivante :

$$\begin{aligned} L_H : \{1, 2, \ldots, C\} \times \mathbb{R} &\to \mathbb{R} \\ y, f(\vec{x}) &\mapsto -\sum_{c=1}^{C} \delta(y, c) \log f_c(\vec{x}) = -\log f_y(\vec{x}). \end{aligned}$$

■

Extension de la fonction d'erreur hinge

Plusieurs propositions permettent de généraliser la fonction d'erreur hinge au cas multi-classe. Dans tous les cas, il s'agit d'assurer que la fonction de décision pour la véritable classe de $\vec{x}$, $f_y(\vec{x})$, prend une valeur supérieure à toutes les autres fonctions de décision $f_c(\vec{x}), c \neq y$.

Weston and Watkins (1999) proposent la définition suivante :

$$L_{\text{hinge}}(y, f(\vec{x})) = \sum_{c \neq y} \left[1 + f_c(\vec{x}) - f_y(\vec{x})\right]_+ .$$

Crammer and Singer (2001), eux, utilisent un maximum plutôt qu'une somme : définition suivante :

$$L_{\text{hinge}}(y, f(\vec{x})) = \left[1 + \max_{c \neq y} f_c(\vec{x}) - f_y(\vec{x})\right]_+ .$$

Ces fonctions de coût sont rarement utilisées en pratique, et on a tendance à leur préférer l'utilisation de la perte hinge binaire dans une approche une-contre-une ou une-contre-toutes.

2.4.3 Coûts pour la régression

Dans le cas d'un problème de régression, nous considérons maintenant $\mathcal{Y} = \mathbb{R}$. Le but de notre fonction de coût est de pénaliser les fonctions de prédiction f dont la valeur est éloignée de la valeur cible $\vec{x}$.

Coût quadratique

Définition 2.17 (Coût quadratique) On appelle *fonction de coût quadratique*, ou *quadratic loss*, ou encore *squared error*, la fonction suivante :

$$\begin{aligned} L_{\text{SE}} : \mathbb{R} \times \mathbb{R} &\to \mathbb{R} \\ y, f(\vec{x}) &\mapsto \frac{1}{2} \left(y - f(\vec{x})\right)^2 . \end{aligned}$$

■

Le coefficient $\frac{1}{2}$ permet d'éviter d'avoir des coefficients multiplicateurs quand on dérive le risque empirique pour le minimiser.

Coût ϵ-insensible

Le coût quadratique a tendance à être dominé par les valeurs aberrantes : dès que quelques observations dans le jeu de données ont une prédiction très éloignée de leur étiquette réelle, la qualité de la prédiction sur les autres observations importe peu. On peut ainsi lui préférer le *coût absolu* :

Définition 2.18 (Coût absolu) On appelle *fonction de coût absolu*, ou *absolute error*, la fonction suivante :

$$\begin{aligned} L_{\text{AE}} : \mathbb{R} \times \mathbb{R} &\to \mathbb{R} \\ y, f(\vec{x}) &\mapsto |y - f(\vec{x})|. \end{aligned}$$

■

Avec cette fonction de coût, même les prédictions très proches de la véritable étiquette sont pénalisées (même si elles le sont faiblement). Cependant, il est numériquement quasiment impossible d'avoir une prédiction exacte. Le coût *ϵ-insensible* permet de remédier à cette limitation.

Définition 2.19 (Coût ϵ-insensible) Étant donné $\epsilon > 0$, on appelle *fonction de coût ϵ-insensible*, ou *ϵ-insensitive loss*, la fonction suivante :

$$\begin{aligned} L_\epsilon : \mathbb{R} \times \mathbb{R} &\to \mathbb{R} \\ y, f(\vec{x}) &\mapsto \max\left(0, |y - f(\vec{x})| - \epsilon\right). \end{aligned}$$

■

Coût de Huber

Le coût ϵ-insensible n'est dérivable ni en $-\epsilon$ ni en $+\epsilon$, ce qui complique l'optimisation du risque empirique. La *fonction de coût de Huber* permet d'établir un bon compromis entre le coût quadratique (dérivable en 0) et le coût absolu (qui n'explose pas dans les valeurs extrêmes).

Définition 2.20 (Coût de Huber) On appelle *fonction de coût de Huber*, ou *Huber loss*, la fonction suivante :

$$\begin{aligned} L_{\text{Huber}} : \mathbb{R} \times \mathbb{R} &\to \mathbb{R} \\ y, f(\vec{x}) &\mapsto \begin{cases} \frac{1}{2}\left(y - f(\vec{x})\right)^2 & \text{si } |y - f(\vec{x})| < \epsilon \\ \epsilon|y - f(\vec{x})| - \frac{1}{2}\epsilon^2 & \text{sinon.} \end{cases} \end{aligned}$$

■

Le terme $-\frac{1}{2}\epsilon^2$ permet d'assurer la continuité de la fonction.

Les fonctions de coût pour la régression sont illustrées sur la figure 2.5.

2.5 Généralisation et sur-apprentissage

2.5.1 Généralisation

Imaginons un algorithme qui, pour prédire l'étiquette d'une observation $\vec{x}$, retourne son étiquette si $\vec{x}$ appartient aux données dont l'étiquette est connue, et une valeur aléatoire sinon. Cet algorithme aura une erreur empirique minimale quelle que soit la fonction de coût choisie, mais fera de très mauvaises prédictions pour toute nouvelle observation. Ce n'est pas vraiment ce que l'on a en tête quand on parle d'*apprentissage*.

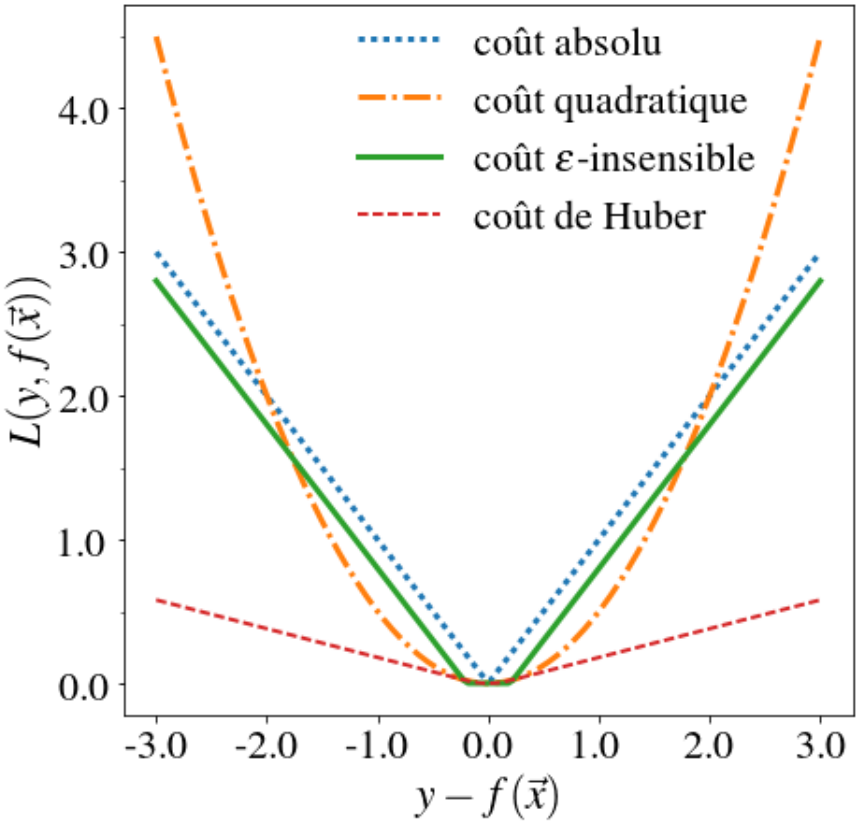


FIGURE 2.5 – Fonctions de coût pour un problème de régression.

Attention

Ainsi, évaluer un algorithme de machine learning sur les données sur lesquelles il a appris ne nous permet absolument pas de savoir comment il se comportera sur de nouvelles données, en d'autres mots, sa capacité de *généralisation*. C'est un point essentiel !

Définition 2.21 (Généralisation) On appelle *généralisation* la capacité d'un modèle à faire des prédictions correctes sur de nouvelles données, qui n'ont pas été utilisées pour le construire. ■

2.5.2 Sur-apprentissage

L'exemple, certes extrême, que nous avons pris plus haut, illustre que l'on peut facilement mettre au point une procédure d'apprentissage qui produise un modèle qui fait de bonnes prédictions sur les données utilisées pour le construire, mais généralise mal. Au lieu de modéliser la vraie nature des objets qui nous intéressent, un tel modèle capture aussi (voire surtout) un bruit qui n'est pas pertinent pour l'application considérée. En effet, dans tout problème d'apprentissage automatique, nos données sont inévitablement bruitées

— par des *erreurs de mesure* dues à la faillibilité des capteurs utilisés pour mesurer les variables par lesquelles on représente nos données, ou à la faillibilité des opérateurs humains qui ont entré ces mesures dans une base de données ;

— par des *erreurs d'étiquetage* (souvent appelés *teacher's noise* en anglais) dues à la faillibilité des opérateurs humains qui ont étiqueté les données ;
— enfin, parce que les variables mesurées ne suffisent pas à modéliser le phénomène qui nous intéresse, soit qu'on ne les connaisse pas, soit qu'elles soient coûteuses à mesurer.

Exemple

Supposons que nous voulions classifier des photographies selon qu'elles représentent des pandas ou non. Chaque image est représentée par les valeurs RGB des pixels qui la composent. Nous aimerions faire en sorte que le modèle que nous construisons capture la véritable nature d'un panda. Nous pouvons cependant être exposés à des erreurs de mesure (erreurs techniques des capteurs de l'appareil photo) ainsi que des erreurs d'étiquetage (erreurs de la personne qui a dû décider, pour chaque photo, s'il s'agissait ou non d'un panda, et a pu cliquer sur le mauvais choix, ou confondre un panda avec un ours). De plus, nous sommes limités par notre choix de variables : nos pixels ne capturent pas directement le fait qu'un panda est un animal rondouillard, avec un masque autour des yeux, généralement entouré de bambous.

Remarque

On voit ici que le choix des variables utilisées pour représenter les données est une étape très importante du processus de modélisation. Nous verrons d'ailleurs au chapitre 7 que la puissance des réseaux de neurones profonds qui sont si populaires de nos jours vient de leur capacité à extraire des données une représentation pertinente. Les techniques présentées dans le chapitre 11 pourront être utilisées pour guider le choix des variables prédictives à utiliser.

Définition 2.22 (Sur-apprentissage) On dit d'un modèle qui, plutôt que de capturer la nature des objets à étiqueter, modélise aussi le bruit et ne sera pas en mesure de généraliser qu'il *sur-apprend*. En anglais, on parle d'*overfitting*. ■

Un modèle qui sur-apprend est généralement un modèle *trop complexe*, qui « colle » trop aux données et capture donc aussi leur bruit.

À l'inverse, il est aussi possible de construire un modèle *trop simple*, dont les performances ne soient bonnes ni sur les données utilisées pour le construire, ni en généralisation.

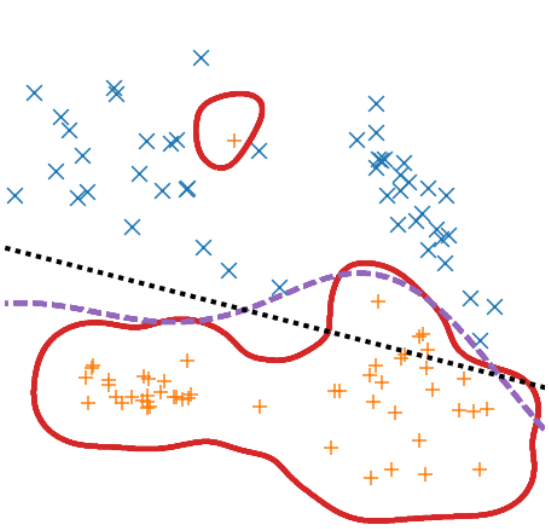
Définition 2.23 (Sous-apprentissage) On dit d'un modèle qui est trop simple pour avoir de bonnes performances même sur les données utilisées pour le construire qu'il *sous-apprend*. En anglais, on parle d'*underfitting*. ■

Ces concepts sont illustrés sur la figure 2.6a pour un problème de classification binaire et la figure 2.6b pour un problème de régression.

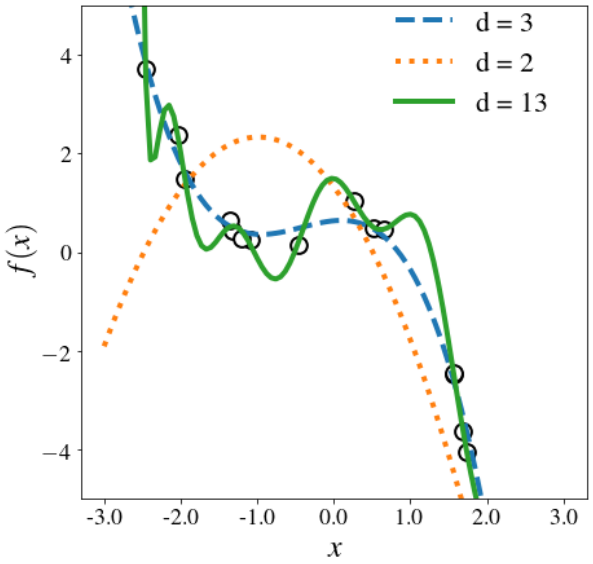
2.5.3 Compromis biais-variance

Pour mieux comprendre le risque d'un modèle $f : \mathcal{X} \to \mathcal{Y}$, nous pouvons le comparer à l'erreur minimale $\mathcal{R}^*$ qui peut être atteinte par n'importe quelle fonction mesurable de $\mathcal{X}$ dans $\mathcal{Y}$: c'est ce qu'on appelle l'*excès d'erreur*, et que l'on peut décomposer de la façon suivante :

$$\mathcal{R}(f) - \mathcal{R}^* = \left[\mathcal{R}(f) - \min_{h \in \mathcal{F}} \mathcal{R}(h)\right] + \left[\min_{h \in \mathcal{F}} \mathcal{R}(h) - \mathcal{R}^*\right] \tag{2.4}$$



(A) Pour séparer les observations négatives (x) des observations positives (+), la droite pointillée sous-apprend. La frontière de séparation en trait plein ne fait aucune erreur sur les données mais est susceptible de sur-apprendre. La frontière de séparation en trait discontinu est un bon compromis.



(B) Les étiquettes y des observations (représentées par des points) ont été générées à partir d'un polynôme de degré $d = 3$. Le modèle de degré $d = 2$ approxime très mal les données et sous-apprend, tandis que celui de degré $d = 13$, dont le risque empirique est plus faible, sur-apprend.

FIGURE 2.6 – Sous-apprentissage et sur-apprentissage

Le premier terme, $\mathcal{R}(f) - \min_{h \in \mathcal{F}} \mathcal{R}(h)$, quantifie la distance entre le modèle f et le modèle optimal sur $\mathcal{F}$. C'est ce que l'on appelle *l'erreur d'estimation.*

Le second terme, $\min_{h \in \mathcal{F}} \mathcal{R}(h) - \mathcal{R}^*$, quantifie la qualité du modèle optimal sur $\mathcal{F}$, autrement dit, la qualité du choix de l'espace des hypothèses. C'est ce que l'on appelle *l'erreur d'approximation.* Si $\mathcal{F}$ est l'ensemble des fonctions mesurables, alors l'erreur d'approximation est nulle.

Ainsi, l'écriture 2.4 permet de décomposer l'erreur entre un terme qui découle de la qualité de l'espace des hypothèses et un autre qui découle de la qualité de la procédure d'optimisation utilisée. En pratique, sauf dans des cas très particuliers où cela est rendu possible par construction, il n'est pas possible de calculer ces termes d'erreur. Cependant, cette écriture nous permet de comprendre le problème suivant : choisir un espace des hypothèses plus large permet généralement de réduire l'erreur d'approximation, car un modèle plus proche de la réalité a plus de chances de se trouver dans cet espace. Cependant, puisque cet espace est plus vaste, la solution optimale y est aussi généralement plus difficile à trouver : l'erreur d'estimation, elle, augmente. C'est dans ce cas qu'il y a sur-apprentissage.

Un espace des hypothèses plus large permet généralement de construire des modèles plus complexes : par exemple, l'ensemble des droites vs. l'ensemble des polynômes de degré 9 (cf. figure 2.6b). C'est une variante du principe du *rasoir d'Ockham*, selon lequel les hypothèses les plus simples sont les plus vraisemblables.

Il y a donc un compromis entre erreur d'approximation et erreur d'estimation : il est difficile de réduire l'une sans augmenter l'autre. Ce compromis est généralement appelé *compromis biais-variance* : l'erreur d'approximation correspond au *biais* de la procédure d'apprentissage, tandis que l'erreur d'estimation correspond à sa *variance*. On retrouvera ce compromis dans l'estimation bayésienne de paramètres à la

section 4.3.3.

Exemple

Considérons pour un problème de régression un espace des hypothèses naïf qui ne contient que des fonctions constantes. Supposons que les étiquettes soient générées par une distribution normale centrée en a. Quelles que soient les données observées, la procédure d'apprentissage va construire un modèle qui retourne a quelle que soit l'observation concernée : la *variance* de la procédure par rapport au jeu de données est très faible. À l'inverse, comme la fonction de prédiction apprise est très peu sensible au jeu de données, il y a un *biais* très important qui conduit à construire des prédicteurs qui retournent a pour toutes les observations.

2.5.4 Régularisation

Plus un modèle est simple, et moins il a de chances de sur-apprendre. Pour limiter le risque de sur-apprentissage, il est donc souhaitable de limiter la complexité d'un modèle. C'est ce que permet de faire la *régularisation*, une technique que nous verrons plus en détail au cours de cet ouvrage (à commencer par le chapitre 6), et qui consiste à ajouter au terme d'erreur que l'on cherche à minimiser un terme qui mesure la complexité du problème (par exemple, dans le cas précédent, le degré du polynôme ou le nombre de coefficients du modèle). Ainsi, un modèle complexe qui a une erreur empirique faible peut être défavorisé face à une modèle plus simple, même si celui-ci présente une erreur empirique plus élevée.

Points clefs

- Les trois ingrédients d'un algorithme d'apprentissage supervisé sont :
  - l'espace des hypothèses,
  - la fonction de coût,
  - l'algorithme d'optimisation qui permet de trouver l'hypothèse optimale au sens de la fonction de coût sur les données (minimisation du risque empirique).
- Le compromis biais-variance traduit le compromis entre l'erreur d'approximation, correspondant au biais de l'algorithme d'apprentissage, et l'erreur d'estimation, correspondant à sa variance.
- La généralisation et le sur-apprentissage sont des préoccupations majeures en machine learning : comment s'assurer que des modèles entraînés pour minimiser leur erreur de prédiction sur les données observées seront généralisables aux données pour lesquelles il nous intéresse de faire des prédictions ?

Pour aller plus loin

- La notion de complexité d'un modèle a été formalisée par Vladimir Vapnik et Alexey Chervonenkis dans les années 1970, et est détaillée par exemple dans l'ouvrage de Vapnik (1995).
- Pour en savoir plus sur la théorie de l'apprentissage, on pourra se référer au livre de Kearns et Vazirani (1994).
- On trouvera une discussion détaillée du compromis biais-variance dans Friedman (1997).

Bibliographie

Crammer, K. and Singer, Y. (2001). On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2 :265–292.

Friedman, J. H. (1997). On bias, variance, 0/1-loss and the curse of dimensionality. *Data Mining and Knowledge Discovery*, 1 :55–77.

Kearns, M. J. et Vazirani, U. V. (1994). *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer, New York.

Weston, J. and Watkins, C. (1999). Support vector machines for multi-class pattern recognition. In *European Symposium on Artificial Neural Networks*.

Chapitre 3

Sélection de modèle et évaluation

Nous avons formalisé au chapitre 2 l'apprentissage supervisé comme la recherche d'un modèle dont l'erreur empirique est minimale sur un ensemble donné d'observations. Cependant, minimiser cette erreur empirique ne garantit pas de minimiser l'erreur du modèle sur la totalité de l'espace des données. En effet, dans une situation de sur-apprentissage, l'erreur du modèle sera sous-estimée. C'est cependant cette erreur, ou, en d'autres mots, notre capacité à faire des prédictions sur des choses qui ne sont pas connues, qui nous intéresse. Ce chapitre présente comment mettre en place un cadre expérimental qui permette d'évaluer un modèle en évitant le biais du sur-apprentissage. Dans cette optique, nous veillerons à distinguer l'*évaluation* d'un modèle, qui consiste à déterminer sa performance sur l'espace des données dans sa totalité, de sa *sélection*, qui consiste à choisir le meilleur modèle parmi plusieurs.

Objectifs

— concevoir un cadre expérimental dans lequel sélectionner un modèle d'apprentissage supervisé ;

— choisir un ou des critères d'évaluation d'un modèle d'apprentissage supervisé ;

— estimer la performance *en généralisation* d'un modèle d'apprentissage supervisé.

Le théorème du *no free lunch* de Wolpert et Macready (1997) indique qu'aucun algorithme de machine learning ne peut bien fonctionner pour *tous* les problèmes d'apprentissage : un algorithme qui fonctionne bien sur un type particulier de problèmes le compensera en fonctionnant moins bien sur d'autres types de problèmes. En d'autres termes, il n'y a pas de « baguette magique » qui puisse résoudre tous nos problèmes de machine learning, et il est donc essentiel, pour un problème donné, de tester plusieurs possibilités afin de sélectionner le modèle optimal. Notons au passage que plusieurs critères peuvent intervenir dans ce choix : non seulement celui de la qualité des prédictions, qui nous intéresse dans ce chapitre, mais aussi celui des ressources de calcul nécessaires, qui peuvent être un facteur limitant en pratique.

3.1 Estimation empirique de l'erreur de généralisation

L'erreur empirique mesurée sur les observations qui ont permis de construire le modèle est un mauvais estimateur de l'erreur du modèle sur l'ensemble des données possibles, ou *erreur de généralisation* : si le modèle sur-apprend, cette erreur empirique peut être proche de zéro voire nulle, tandis que l'erreur de généralisation peut être arbitrairement grande.

3.1.1 Jeu de test

Il est donc indispensable d'utiliser pour évaluer un modèle des données étiquetées qui n'ont pas servi à le construire. La manière la plus simple d'y parvenir est de mettre de côté une partie des observations, réservées à l'évaluation du modèle, et d'utiliser uniquement le reste des données pour le construire.

Définition 3.1 (Jeu de test, jeu d'entraînement) Étant donné un jeu de données $\mathcal{D} = \{(\vec{x}^{\,i}, y^i)\}_{i=1,\dots,n}$, partitionné en deux jeux $\mathcal{D}_{\text{tr}}$ et $\mathcal{D}_{\text{te}}$, on appelle *jeu d'entraînement* (*training set* en anglais) l'ensemble $\mathcal{D}_{\text{tr}}$ utilisé pour entraîner un modèle prédictif, et *jeu de test* (*test set* en anglais) l'ensemble $\mathcal{D}_{\text{te}}$ utilisé pour son évaluation. ■

Comme nous n'avons pas utilisé le jeu de test pour entraîner notre modèle, il peut être considéré comme un jeu de données « nouvelles ». La perte calculée sur ce jeu de test est un estimateur de l'erreur de généralisation.

3.1.2 Jeu de validation

Considérons maintenant la situation dans laquelle nous voulons choisir entre K modèles. Nous pouvons alors entraîner chacun des modèles sur le jeu de données d'entraînement, obtenant ainsi K fonctions de décision $f_1, f_2, \dots, f_K$, puis calculer l'erreur de chacun de ces modèles sur le jeu de test. Nous pouvons ensuite choisir comme modèle celui qui a la plus petite erreur sur le jeu de test :

$$\hat{f} = \underset{k=1,\dots,K}{\arg\min} \frac{1}{|\mathcal{D}_{\text{te}}|} \sum_{\vec{x},y \in \mathcal{D}_{\text{te}}} L(y, f_k(\vec{x})) \tag{3.1}$$

Mais quelle est son erreur de généralisation ? Comme nous avons utilisé $\mathcal{D}_{\text{te}}$ pour *sélectionner* le modèle, il ne représente plus un jeu indépendant composé de données nouvelles, inutilisées pour déterminer le modèle.

La solution est alors de découper notre jeu de données en *trois* parties :

— Un *jeu d'entraînement* $\mathcal{D}_{\text{tr}}$ sur lequel nous pourrons entraîner nos K algorithmes d'apprentissage ;
— Un *jeu de validation* (*validation set* en anglais) $\mathcal{D}_{\text{val}}$ sur lequel nous évaluerons les K modèles ainsi obtenus, afin de *sélectionner* un modèle définitif ;
— Un *jeu de test* $\mathcal{D}_{\text{te}}$ sur lequel nous évaluerons enfin l'erreur de généralisation du modèle choisi.

On voit ici qu'il est important de distinguer la *sélection* d'un modèle de son *évaluation* : les faire sur les mêmes données peut nous conduire à sous-estimer l'erreur de généralisation et le sur-apprentissage du modèle choisi.

Remarque

Une fois un modèle sélectionné, on peut le ré-entraîner sur l'union du jeu d'entraînement et du jeu de validation afin de construire un modèle final.

3.1.3 Validation croisée

La séparation d'un jeu de données en un jeu d'entraînement et un jeu de test est nécessairement arbitraire. Nous risquons ainsi d'avoir, par hasard, créé des jeux de données qui ne sont pas représentatifs. Pour éviter cet écueil, il est souhaitable de reproduire plusieurs fois la procédure, puis de moyenner les résultats obtenus afin de moyenner ces effets aléatoires. Le cadre le plus classique pour ce faire est celui de la *validation croisée*, illustré sur la figure 3.1

Définition 3.2 (Validation croisée) Étant donné un jeu $\mathcal{D}$ de n observations, et un nombre K, on appelle *validation croisée* la procédure qui consiste à

1. partitionner $\mathcal{D}$ en K parties de tailles sensiblement similaires, $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_K$
2. pour chaque valeur de $k = 1, \ldots, K$,
   — entraîner un modèle sur $\bigcup_{l \neq k} \mathcal{D}_l$
   — évaluer ce modèle sur $\mathcal{D}_k$.

Chaque partition de $\mathcal{D}$ en deux ensembles $\mathcal{D}_k$ et $\bigcup_{l \neq k} \mathcal{D}_l$ est appelée un *fold* de la validation croisée. ■

Chaque observation étiquetée du jeu $\mathcal{D}$ appartient à un unique jeu de test, et à $(K-1)$ jeux d'entraînement. Ainsi, cette procédure génère une prédiction par observation de $\mathcal{D}$. Pour conclure sur la performance du modèle, on peut :

— soit évaluer la qualité des prédictions sur $\mathcal{D}$;
— soit évaluer la qualité de chacun des K prédicteurs sur le jeu de test $\mathcal{D}_k$ correspondant, et moyenner leurs performances. Cette deuxième approche permet aussi de rapporter l'écart-type de ces performances, ce qui permet de se faire une meilleure idée de la variabilité de la qualité des prédictions en fonction des données d'entraînement.

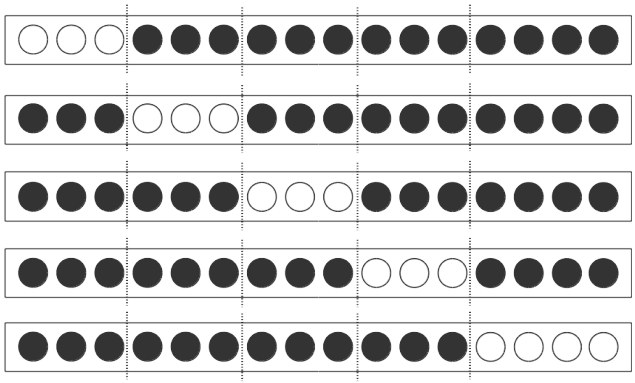


FIGURE 3.1 – Une validation croisée en 5 *folds* : Chaque observation appartient à un des 5 jeux de validation (en blanc) et aux 4 autres jeux d'entraînement (en noir).

Stratification

Définition 3.3 (Validation croisée stratifiée) Une validation croisée est dite *stratifiée* si la moyenne des étiquettes des observations est sensiblement la même dans chacun des K sous-ensembles $\mathcal{D}_k$:

$$\frac{1}{|\mathcal{D}_1|} \sum_{i \in \mathcal{D}_1} y^i \approx \frac{1}{|\mathcal{D}_2|} \sum_{i \in \mathcal{D}_2} y^i \approx \cdots \approx \frac{1}{|\mathcal{D}_K|} \sum_{i \in \mathcal{D}_K} y^i \approx \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} y^i.$$

■

Dans le cas d'un problème de classification, cela signifie que la proportion d'exemples de chaque classe est la même dans chacun des $\mathcal{D}_k$. Cette proportion est donc aussi la même que dans le jeu de données $\mathcal{D}$ complet.

L'intérêt de cette procédure est de faire en sorte que la distribution des observations au sein de chaque $\mathcal{D}_k$ soit la même qu'au sein du jeu de données $\mathcal{D}$. Imaginons que par malchance un des folds ne contienne que des exemples positifs dans son jeu d'entraînement et que des exemples négatifs dans son jeu de test : il est vraisemblable que, sur ce fold, tout modèle apprenne à prédire que tout est positif et ait une très mauvaise performance.

Leave-one-out

Un algorithme d'apprentissage apprendra d'autant mieux qu'il y a d'avantage de données disponibles pour l'entraînement : plus on connaît d'étiquettes pour des observations de l'espace $\mathcal{X}$, plus on peut contraindre le modèle à les respecter. Or pour un jeu de données de taille n, un jeu de test d'une validation croisée à K folds contient $\frac{(K-1)n}{K}$ points : les modèles entraînés apprendront d'autant mieux sur chacun des folds qu'ils sont grands, ce qui nous pousse à considérer le cas où $K = n$.

Définition 3.4 (Validation croisée leave-one-out) Une validation croisée dont le nombre de folds est égal au nombre d'observations dans le jeu d'entraînement, et dont chaque fold est donc composé d'un jeu d'entraînement de taille $n - 1$ et d'un jeu de test de taille 1, est appelée *leave one out* : on met de côté, pour chaque fold, un unique exemple. ■

L'évaluation par leave-one-out présente deux inconvénients. Tout d'abord, elle requiert un grand temps de calcul : on entraîne n modèles, chacun sur $n-1$ observations, au lieu de (dans le cas $K = 10$) 10 modèles, chacun sur 90% des observations. De plus, les jeux d'entraînement ainsi formés sont très similaires entre eux. Les modèles entraînés seront eux aussi très similaires, et généralement peu différents d'un modèle entraîné sur l'intégralité du jeu de données. Par contre, les jeux de test seront disjoints, et les performances pourront ainsi avoir une grande variabilité, ce qui compliquera leur interprétation.

3.1.4 Bootstrap

Une autre façon de rééchantillonner les données afin d'estimer l'erreur de généralisation est connue sous le nom de *bootstrap*.

Définition 3.5 (Bootstrap) Étant donné un jeu $\mathcal{D}$ de n observations, et un nombre B, on appelle *bootstrap* la procédure qui consiste à créer B échantillons $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_B$ de $\mathcal{D}$, obtenus chacun en tirant n exemples de $\mathcal{D}$ *avec remplacement*. Ainsi, chaque exemple peut apparaître plusieurs fois, ou pas du tout, dans $\mathcal{D}_b$. ■

Le bootstrap est une procédure couramment utilisée en statistiques pour estimer un paramètre en fonction de son estimation sur les B échantillons. En la suivant, on pourrait entraîner le modèle à évaluer sur chaque échantillon $\mathcal{D}_b$, puis évaluer sa performance sur l'intégralité de $\mathcal{D}$. Cependant, cette estimation serait biaisée par la présence d'une partie des exemples de $\mathcal{D}$ dans $\mathcal{D}_b$. Il faut donc se limiter aux exemples de $\mathcal{D} \setminus \mathcal{D}_b$. En pratique, cette procédure est jugée trop complexe pour être souvent appliquée.

Remarque

La probabilité que $(\vec{x}^{\,i}, y^i)$ apparaisse dans $\mathcal{D}_b$ peut être calculée comme le complémentaire à 1 de la probabilité que $(\vec{x}^{\,i}, y^i)$ ne soit tiré aucune des n fois. La probabilité de $(\vec{x}^{\,i}, y^i)$ soit tiré une fois vaut $\frac{1}{n}$. Ainsi

$$\mathbb{P}[(\vec{x}^{\,i}, y^i) \in \mathcal{D}_b] = 1 - \left(1 - \frac{1}{n}\right)^n.$$

Quand n est grand, cette probabilité vaut donc environ $1 - e^{-1} \approx 0.632$, car la limite en $+\infty$ de $\left(1 + \frac{x}{n}\right)^n$ vaut e^x.

Ainsi, $\mathcal{D}_b$ contient environ deux tiers des observations de $\mathcal{D}$.

3.2 Critères de performance

Il existe de nombreuses façons d'évaluer la performance prédictive d'un modèle d'apprentissage supervisé. Cette section présente les principaux critères utilisés.

3.2.1 Matrice de confusion et critères dérivés

Comme nous l'avons vu, le nombre d'erreurs de classification permet d'évaluer la qualité d'un modèle prédictif. Notons que l'on préférera généralement décrire le nombre d'erreurs comme une fraction du nombre d'exemples : un taux d'erreur de 1% est plus parlant qu'un nombre absolu d'erreurs.

Mais toutes les erreurs ne se valent pas nécessairement. Prenons l'exemple d'un modèle qui prédise si oui ou non une radiographie présente une tumeur inquiétante : une fausse alerte, qui sera ensuite infirmée par des examens complémentaires, est moins problématique que de ne pas déceler la tumeur et de ne pas traiter la personne concernée. Les performances d'un modèle de classification, binaire comme multi-classe, peuvent être résumée dans une *matrice de confusion*.

Définition 3.6 (Matrice de confusion) Étant donné un problème de classification, on appelle *matrice de confusion* une matrice M contenant autant de lignes que de colonnes que de classes, et dont l'entrée M_{ck} est le nombre d'exemples de la classe c pour laquelle l'étiquette k a été prédite.

Dans le cas de la classification binaire, la matrice de confusion prend la forme suivante :

		Classe réelle	
		0	1
Classe prédite	0	vrais négatifs (TN)	faux négatifs (FN)
	1	faux positifs (FP)	vrais positifs (TP)

On appelle *vrais positifs* (en anglais *true positives*) les exemples positifs correctement classifiés ; *faux positifs* (en anglais *false positives*) les exemples négatifs étiquetés positifs par le modèle ; et réciproquement pour les *vrais négatifs* (*true negatives*) et les *faux négatifs* (*false negatives*). On note généralement par *TP* le nombre de vrais positifs, *FP* le nombre de faux positifs, *TN* le nombre de vrais négatifs et *FN* le nombre de faux négatifs.

Les faux positifs sont aussi appelés *fausses alarmes* ou *erreurs de type I*, par opposition aux *erreurs de type II* qui sont les faux négatifs. ■

Il est possible de dériver de nombreux critères d'évaluation à partir de la matrice de confusion. En voici quelques exemples :

Définition 3.7 (Rappel) On appelle *rappel* (*recall* en anglais), ou *sensibilité* (*sensitivity* en anglais), le taux de vrais positifs, c'est-à-dire la proportion d'exemples positifs correctement identifiés comme tels :

$$\text{Rappel} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

■

Il est cependant très facile d'avoir un bon rappel en prédisant que *tous* les exemples sont positifs. Ainsi, ce critère ne peut pas être utilisé seul. On lui adjoint ainsi souvent la *précision* :

Définition 3.8 (Précision) On appelle *précision*, ou *valeur positive prédictive* (*positive predictive value, PPV*) la proportion de prédictions correctes parmi les prédictions positives :

$$\text{Précision} = \frac{\text{TP}}{\text{TP} + \text{FP}}.$$

■

De même que l'on peut facilement avoir un très bon rappel au détriment de la précision, il est aisé d'obtenir une bonne précision (au détriment du rappel) en faisant très peu de prédictions positives (ce qui réduit le risque qu'elles soient erronées)

Attention

L'anglais distingue *precision* (la précision ci-dessus) et *accuracy*, qui est la proportion d'exemples correctement étiquetés, soit le complémentaire à 1 du taux d'erreur, aussi traduit par *précision* en français. On utilisera donc ces termes avec précaution.

Pour résumer rappel et précision en un seul nombre, on calculera la *F-mesure* :

Définition 3.9 (F-mesure) On appelle *F-mesure* (*F-score* ou *F1-score* en anglais) la moyenne harmonique de la précision et du rappel :

$$F = 2\frac{\text{Précision . Rappel}}{\text{Précision} + \text{Rappel}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}.$$

■

Définition 3.10 (Spécificité) On appelle *spécificité* le taux de vrais négatifs, autrement dit la proportion d'exemples négatifs correctement identifiés comme tels.

$$\text{Spécificité} = \frac{\text{TN}}{\text{FP} + \text{TN}}.$$

■

Exemple

Prenons l'exemple d'un test clinique pour illustrer ces différents critères. Il ne s'agit pas ici d'un modèle d'apprentissage automatique, mais d'un frottis de dépistage du cancer du col de l'utérus : il s'agit d'un examen beaucoup plus simple et moins invasif qu'un examen histologique, qui doit être interprété par un expert, et servira de vérité terrain.

Les résultats d'une expérience menée sur $4\,000$ femmes âgées de 40 ans et plus sont présentés sur le tableau 3.1.

	Cancer	Pas de cancer	Total
Frottis +	190	210	400
Frottis -	10	3590	3600
Total	200	3800	4000

TABLE 3.1 – Matrice de confusion pour une expérience sur le dépistage du cancer du col de l'utérus par frottis.

Le rappel est de 95%, la spécificité de 94.5%, mais la précision ne vaut que 47.5%. En effet, ce test est un bon outil de dépistage : la probabilité de n'avoir effectivement pas de cancer quand le frottis est négatif est élevée ($3590/3600 \approx 99.7\%$). Cependant, c'est un mauvais outil diagnostique, au sens où la probabilité de fausse alarme est très élevée.

3.2.2 Évaluation de méthodes de classification binaire retournant un score

De nombreux algorithmes de classification ne retournent pas directement une étiquette de classe, mais utilisent une fonction de décision qui doit ensuite être seuillée pour devenir une étiquette. Cette fonction de décision peut être un score arbitraire (par exemple, la proportion d'exemples positifs parmi les k plus proches voisins du point à étiqueter – nous verrons l'algorithme des k plus proches voisins en détails au chapitre 8) ou la probabilité d'appartenir à la classe positive (comme c'est par exemple le cas pour la régression logistique présentée à la section 5.3).

Plusieurs critères permettent d'évaluer la qualité de la fonction de décision avant seuillage.

Courbe ROC

Définition 3.11 (Courbe ROC) On appelle *courbe ROC*, de l'anglais *Receiver-Operator Characteristic* la courbe décrivant l'évolution de la sensibilité en fonction du complémentaire à 1 de la spécificité, parfois appelé *antispécificité*, lorsque le seuil de décision change.

Le terme vient des télécommunications, où ces courbes servent à étudier si un système arrive à séparer le signal du bruit de fond.

On peut synthétiser une courbe ROC par l'aire sous cette courbe, souvent abrégée *AUROC* pour *Area Under the ROC*. ■

Un exemple de courbe ROC est présenté sur la figure 3.2. Le point $(0, 0)$ apparaît quand on utilise comme seuil un nombre supérieur à la plus grande valeur retournée par la fonction de décision : ainsi, tous les exemples sont étiquetés négatifs. À l'inverse, le point $(1, 1)$ apparaît quand on utilise pour seuil une valeur inférieure au plus petit score retourné par la fonction de décision : tous les exemples sont alors étiquetés positifs.

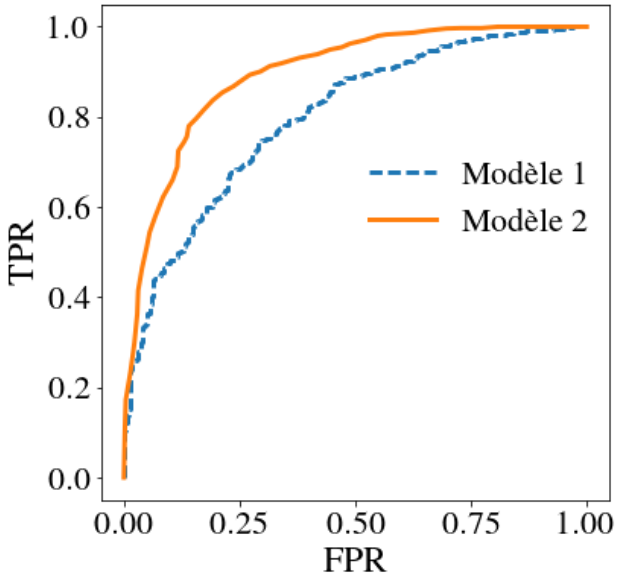


FIGURE 3.2 – Les courbes ROC de deux modèles.

Pour construire la courbe ROC, on prend pour seuil les valeurs successives de la fonction de décision sur notre jeu de données. Ainsi, à chaque nouvelle valeur de seuil, une observation que l'on prédisait précédemment négative change d'étiquette. Si cette observation est effectivement positive, la sensibilité augmente

de $1/n_p$ (où n_p est le nombre d'exemples positifs) ; sinon, c'est l'antispécificité qui augmente de $1/n_n$, où n_n est le nombre d'exemples négatifs. La courbe ROC est donc une courbe en escaliers.

Un classifieur idéal, qui ne commet aucune erreur, associe systématique des scores plus faibles aux exemples négatifs qu'aux exemples positifs. Sa courbe ROC suit donc le coin supérieur gauche du carré $[0, 1]^2$; il a une aire sous la courbe de 1.

La courbe ROC d'un classifieur aléatoire, qui fera sensiblement la même proportion d'erreurs que de classifications correctes quel que soit le seuil utilisé, suit la diagonale de ce carré. L'aire sous la courbe ROC d'un classifieur aléatoire vaut donc 0.5.

Exemple

Pour illustrer la construction d'une courbe ROC, prenons l'exemple décrit sur le tableau 3.2.

Étiquette	+	-	+	+	-	-
Score	0.9	0.8	0.6	0.4	0.3	0.1

TABLE 3.2 – Exemple de résultats d'une expérience de classification binaire, évaluée sur 6 échantillons.

Pour un seuil supérieur à 0.9, les 6 exemples sont étiquetés négatifs. On commence donc par le point $(0, 0)$. Pour un seuil entre 0.95 et 0.9, seule la première observation est étiquetée positive. La sensibilité est donc de $\frac{1}{3}$ tandis que l'antispécificité reste nulle. On peut continuer ainsi jusqu'à utiliser un seuil inférieur à 0.1 :

Seuil	> 0.9	0.8–0.9	0.6–0.8	0.4–0.6	0.3–0.4	0.1–0.3	< 0.1
TP/P	0	1/3	1/3	2/3	1	1	1
FP/P	0	0	1/3	1/3	1/3	2/3	1

La courbe ROC correspondante est visible sur la figure 3.3.

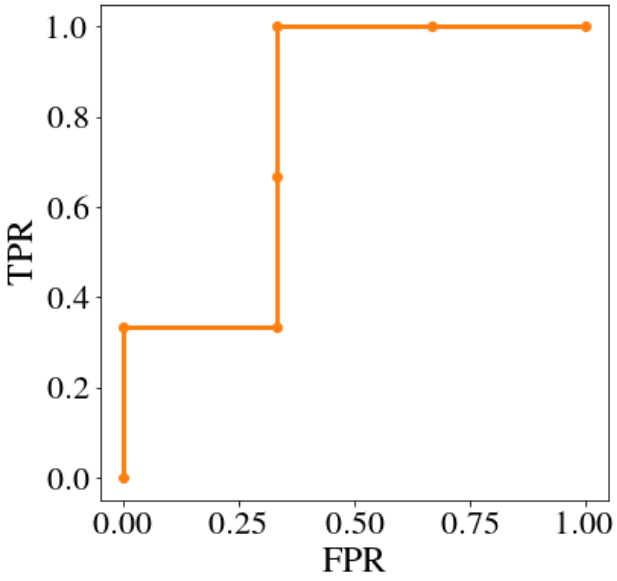


FIGURE 3.3 – Courbe ROC correspondant à l'expérience du tableau 3.2.

On peut enfin utiliser la courbe ROC pour choisir un seuil de décision, à partir de la sensibilité (ou de la spécificité) que l'on souhaite garantir.

Courbe précision-rappel

La courbe *précision-rappel* vient souvent complémenter la courbe ROC.

Définition 3.12 (Courbe précisioj-rappel) On appelle *courbe précision-rappel*, ou *Precision-Recall curve* en anglais, la courbe décrivant l'évolution de la précision en fonction du rappel, lorsque le seuil de décision change.

Pour synthétiser cette courbe, on peut utiliser l'aire sous celle-ci, souvent abrégée *AUPR* pour *Area Under the Precision-Recall curve*. ■

Un exemple de courbe précision-rappel, pour les mêmes données que la figure 3.2, est présenté sur la figure 3.4.

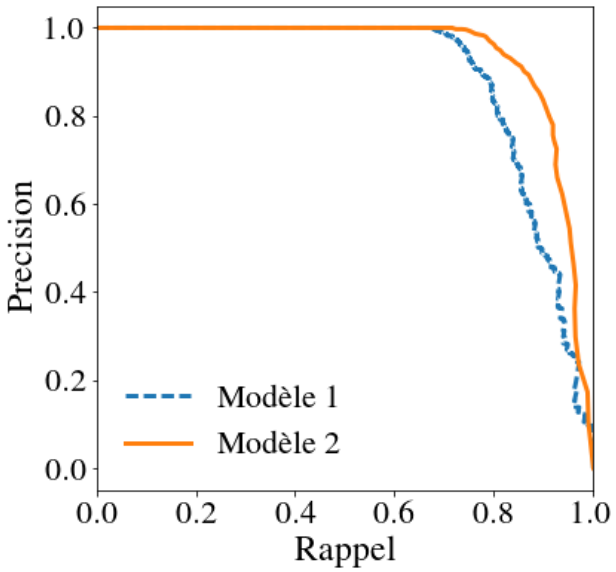


FIGURE 3.4 – Les courbes précision-rappel de deux modèles.

Remarque

Pour le seuil le plus élevé, aucun exemple n'est étiqueté positif, et la précision n'est donc pas définie. Par convention, on utilise généralement une précision de 1 si la première observation à considérer est positive, et une précision de 0 sinon.

Exemple

Reprenons l'exemple précédent pour construire une courbe précision-rappel. Les valeurs de la précision et du rappel sont les suivantes :

Seuil	> 0.9	0.8–0.9	0.6–0.8	0.4–0.6	0.3–0.4	0.1–0.3	< 0.1
Rappel	0	1/3	1/3	2/3	1	1	1
Précision	–	1	1/2	2/3	3/4	3/5	3/6

On obtient donc la courbe précision-rappel visible sur la figure 3.5.

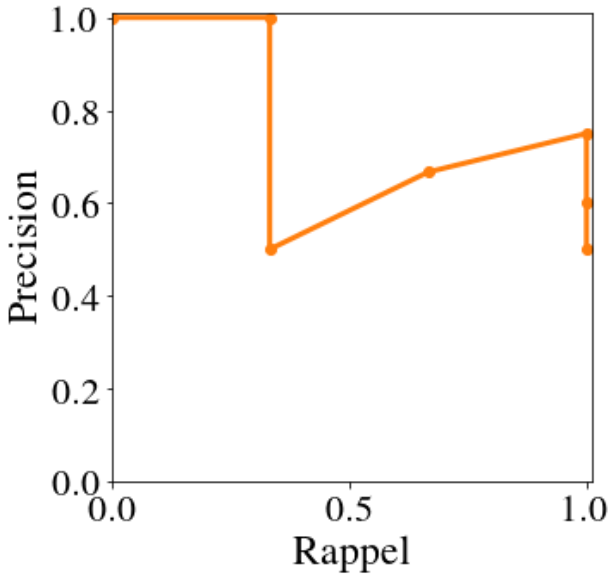


FIGURE 3.5 – Courbe précision-rappel correspondant à l'expérience du tableau 3.2.

3.2.3 Erreurs de régression

Dans le cas d'un problème de régression, le nombre d'erreurs n'est pas un critère approprié pour évaluer la performance. D'une part, à cause des imprécisions numériques, il est délicat de dire d'une prédiction à valeur réelle si elle est correcte ou non. D'autre part, un modèle dont 50% des prédictions sont correctes à 0.1% près et les 50 autres pourcent sont très éloignées des vraies valeurs vaut-il mieux qu'un modèle qui n'est correct qu'à 1% près, mais pour 100% des exemples ?

Ainsi, on préférera quantifier la performance d'un modèle de régression en fonction de l'écart entre les prédictions et les valeurs réelles.

Un premier critère est donc l'erreur quadratique moyenne :

Définition 3.13 (Erreur quadratique moyenne (MSE)) Étant données n étiquettes réelles $y^1, y^2, \dots, y^n$ et n prédictions $f(\vec{x}^{\,1}), f(\vec{x}^{\,2}), \dots, f(\vec{x}^{\,n})$, on appelle *erreur quadratique moyenne*, ou *MSE* de l'anglais *mean squared error* la valeur

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(f(\vec{x}^{\,i}) - y^i\right)^2.$$

■

Pour mesurer l'erreur dans la même unité que la cible, on lui préfère souvent sa racine :

Définition 3.14 (RMSE) Étant données n étiquettes réelles $y^1, y^2, \dots, y^n$ et n prédictions $f(\vec{x}^{\,1}), f(\vec{x}^{\,2}), \dots, f(\vec{x}^{\,n})$, on appelle *racine de l'erreur quadratique moyenne*, ou *RMSE* de l'anglais *root mean squared error* la valeur

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(f(\vec{x}^{\,i}) - y^i)^2}.$$

■

Dans le cas où les valeurs cibles couvrent plusieurs ordres de grandeur, on préfère parfois passer au log avant de comparer $f(\vec{x}^{\,i})$ à y^i, afin de ne pas donner plus d'importance aux erreurs faites pour des valeurs plus élevées.

Définition 3.15 (RMSLE) Étant données n étiquettes réelles $y^1, y^2, \ldots, y^n$ et n prédictions $f(\vec{x}^{\,1}), f(\vec{x}^{\,2}), \ldots, f(\vec{x}^{\,n})$, on appelle *racine du log de l'erreur quadratique moyenne*, ou *RMSLE* de l'anglais *root mean squared log error* la valeur

$$\text{RMSLE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\log\left(f(\vec{x}^{\,i})+1\right)-\log\left(y^i+1\right)\right)^2}.$$

■

L'interprétation de ces erreurs requiert néanmoins de connaître la distribution des valeurs cibles : une RMSE de 1 cm n'aura pas la même signification selon qu'on essaie de prédire la taille d'humains ou celle de drosophiles. Pour répondre à cela, il est possible de normaliser la somme des carrés des résidus non pas en en faisant la moyenne, mais en la comparant à la somme des distances des valeurs cibles à leur moyenne.

Définition 3.16 (Coefficient de détermination) Étant données n étiquettes réelles $y^1, y^2, \ldots, y^n$ et n prédictions $f(\vec{x}^{\,1}), f(\vec{x}^{\,2}), \ldots, f(\vec{x}^{\,n})$, on appelle *erreur carrée relative*, ou *RSE* de l'anglais *relative squared error* la valeur

$$\text{RSE} = \frac{\sum_{i=1}^{n}\left(f(\vec{x}^{\,i})-y^i\right)^2}{\sum_{i=1}^{n}\left(y^i-\frac{1}{n}\sum_{l=1}^{n}y^l\right)^2}.$$

Le complémentaire à 1 de la RSE est le *coefficient de détermination*, noté R^2. ■

On note le coefficient de détermination R^2 car il s'agit du carré du coefficient de corrélation entre $\vec{y}$ et $(f(\vec{x}^{\,1}), f(\vec{x}^{\,2}), \ldots, f(\vec{x}^{\,n}))$ donné par

$$R = \frac{\sum_{i=1}^{n}\left(y^i-\frac{1}{n}\sum_{l=1}^{n}y^l\right)\left(f(\vec{x}^{\,i})-\frac{1}{n}\sum_{l=1}^{n}f(\vec{x}^{\,l})\right)}{\sqrt{\sum_{i=1}^{n}\left(y^i-\frac{1}{n}\sum_{l=1}^{n}y^l\right)^2}\sqrt{\sum_{i=1}^{n}\left(f(\vec{x}^{\,i})-\frac{1}{n}\sum_{l=1}^{n}f(\vec{x}^{\,l})\right)^2}}. \tag{3.2}$$

Ce coefficient indique à quel point les valeurs prédites sont corrélées aux valeurs réelles ; attention, il sera élevé aussi si elles leur sont anti-corrélées.

3.2.4 Comparaison à des algorithmes naïfs

Pour construire un modèle de machine learning, nous nous appuyons d'une part sur les données, et d'autre part sur des hypothèses quant à la forme de ce modèle ; ces hypothèses déterminent l'espace des hypothèses. La validité de ces hypothèses dépend du problème étudié. Ce problème peut être plus ou moins facile, et la performance d'un modèle que nous avons entraîné ne peut être interprétée qu'à la lueur de cette difficulté.

Pour la déterminer, il peut être très utile d'utiliser des approches d'apprentissage naïves, autrement dit très simples, qui utilisent certaines propriétés du jeu d'entraînement mais pas de l'observation à étiqueter. Nous n'attendons pas de bonnes performances de ces méthodes mais elles servent d'étalon pour mieux comprendre les performances mesurées par ailleurs, en nous indiquant le « strict minimum » que l'on peut attendre de nos modèles.

Méthodes naïves pour la classification

Pour un problème de classification, on peut par exemple considérer une des approches suivantes :
— Prédire systématiquement l'étiquette majoritaire dans le jeu d'entraînement.
— Prédire une étiquette aléatoire, selon la distribution des étiquettes dans le jeu d'entraînement.
— Dans le cas d'une classification binaire, prédire des scores de manière uniforme avant de les seuiller. Cette méthode est plus recommandée si l'on cherche à tracer des courbes ROC ou PR.

Remarque

Si le jeu d'entraînement est *déséquilibré*, à savoir qu'une classe y est largement plus présente que les autres, le premier algorithme naïf que nous décrivons peut avoir un taux d'erreur très faible. Il faudra aussi prendre en compte la spécificité de l'algorithme.

Méthodes naïves pour la régression

Pour un problème de régression, on peut considérer les approches naïves suivantes :
— Prédire une valeur aléatoire, uniformément entre la plus petite et la plus grande des valeurs des étiquettes du jeu d'entraînement ;
— Prédire systématiquement la moyenne ou la médiane des étiquettes du jeu d'entraînement.

Points clefs

— Pour éviter le sur-apprentissage, il est essentiel lors de l'étape de sélection du modèle de valider les différents modèles testés sur un jeu de données différent de celui utilisé pour l'entraînement.
— Pour estimer la performance en généralisation d'un modèle, il est essentiel de l'évaluer sur des données qui n'ont été utilisées ni pour l'entraînement, ni pour la sélection de ce modèle.
— De nombreux critères permettent d'évaluer la performance prédictive d'un modèle. On les choisira en fonction de l'application.
— Pour interpréter la performance d'un modèle, il peut être utile de le comparer à une approche naïve.

Pour aller plus loin

- En construisant la courbe précision-rappel de la figure 3.5, nous avons relié les points par des segments. Cette interpolation linéaire n'est en fait pas appropriée. On pourra se référer aux articles de Davis et Goadrich (2006) et Fawcett (2006).
- Pour certains modèles, en particulier linéaires (voir chapitre 5), il est possible d'estimer l'optimisme, c'est-à-dire la différence entre l'erreur d'entraînement et l'erreur de test, de manière théorique, plutôt que d'utiliser une validation croisée. Dans le cas des modèles linéaires, on pourra notamment utiliser le *coefficient Cp de Mallow*, le *critère d'information d'Akaike* ou *le critère d'information bayésien*. Pour plus de détails, on se reportera au livre de Dodge et Rousson (2004).
- La *longueur de description minimale* (ou *minimum description length, MDL*) est un concept issu de la théorie de l'information qui peut être utilisé pour la sélection de modèle (Rissanen, 1978). Dans ce cadre, les étiquettes d'un jeu de données $\mathcal{D}$ peuvent être représentées par, d'une part une représentation d'un modèle, et d'autre part une représentation de la différence entre les prédictions du modèle et

leurs vraies étiquettes dans $\mathcal{D}$. Le meilleur modèle est celui pour lequel la somme des tailles de ces représentations est minimale : un bon modèle permet de compresser efficacement les données.

- Pour une discussion détaillée des procédures de validation croisée, on pourra consulter Arlot et Celisse (2010).

Bibliographie

Arlot, S. et Celisse, A. (2010). A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4 :40–79.

Davis, J. et Goadrich, M. (2006). The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 233–240, New York, NY, USA. ACM.

Dodge, Y. et Rousson, V. (2004). *Analyse de régression appliquée*. Dunod.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, 27 :861–874.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14(5) :465–471.

Wolpert, D. H. et Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1) :67–82.

Chapitre 4

Inférence bayésienne

Un des problèmes fondamentaux auxquels nous faisons face dans le cadre du machine learning est celui de l'incertitude : notre compréhension du monde est limitée par nos observations nécessairement partielles. Les modèles probabilistes nous permettent de prendre en compte cette incertitude de manière explicite.

Dans ce chapitre, nous allons voir comment formuler l'apprentissage d'un modèle comme un problème d'inférence sur une distribution jointe des observations et des paramètres de ce modèle. Nous verrons aussi comment utiliser le cadre bayésien pour prendre des décisions, ce qui requiert de modéliser additionnellement à quel point il est utile de prendre la bonne décision. À titre illustratif, nous présenterons la classification bayésienne naïve.

Objectifs

— Formaliser le concept de classe grâce à des modèles probabilistes ;

— Définir des règles de décision, sur la base de tests de rapport de vraisemblance ;

— Estimer une densité par maximum de vraisemblance ou par estimateur de Bayes ;

— Mettre en œuvre un algorithme de classification naïve bayésienne.

4.1 Modèles génératifs pour la classification binaire

L'approche statistique de la classification formalise le concept de classe grâce à des modèles probabilistes. Dans ce cadre, nous allons supposer que les n observations $\vec{x}^{\,1}, \vec{x}^{\,2}, \ldots, \vec{x}^{\,n}$ sont la réalisation d'une variable aléatoire $X \in \mathcal{X}$. De plus, nous supposons que leurs étiquettes $y^1, y^2, \ldots, y^n$ sont la réalisation d'une variable aléatoire $Y \in \{0, 1\}$. Dans le cas de la classificaton multi-classe, nous utiliserons une valeur aléatoire $Y \in \{1, 2, \ldots, C\}$ (où C est le nombre total de classes).

Considérons maintenant une loi de probabilité jointe $\mathbb{P}(X, Y)$ pour l'ensemble des variables entrant dans le modèle, autrement dit X et Y. Cette approche est qualifiée de *modélisation générative* : elle répond à la question « Comment les données que l'on observe auraient-elles pu être générées ? ». Bien sûr, en pratique, nos données ne sont pas toujours le fruit d'un processus aléatoire ; cependant, considérer qu'elles

sont la réalisation d'une variable aléatoire peut être une façon efficace de représenter l'information complexe qu'elles contiennent. Ainsi, si l'une de nos variables est le résultat d'un test de dépistage médical, il est plus simple de la représenter comme la réalisation d'une loi de Bernoulli que de modéliser toutes les informations biochimiques qui ont pu aboutir à ce résultat. C'est dans ce cadre que nous allons maintenant nous placer.

4.1.1 Inférence et prédiction

La probabilité qu'une observation $\vec{x}$ appartienne à la classe c est ainsi déterminée par $\mathbb{P}(Y = c|X = \vec{x})$. Deux problèmes se posent alors :

- Un problème d'*inférence*, qui consiste à déterminer les lois de probabilité $\mathbb{P}(Y = c|X = \vec{x})$ à partir de nos observations et hypothèses ;
- Un problème de *prédiction*, ou de *décision*, qui consiste à utiliser ces lois pour déterminer la classe d'une observation $\vec{x}$.

Pour *prédire* la classe d'une observation $\vec{x}$, il semble raisonnable de déterminer la classe la plus probable étant donnée cette observation. Il s'agit alors de comparer $\mathbb{P}(Y = 1|X = \vec{x})$ et $\mathbb{P}(Y = 0|X = \vec{x})$. Formellement, nous considérons alors la *règle de décision* suivante :

$$\hat{y} = \begin{cases} 1 & \text{si } \mathbb{P}(Y = 1|X = \vec{x}) > \mathbb{P}(Y = 0|X = \vec{x}) \\ 0 & \text{sinon.} \end{cases} \tag{4.1}$$

Pour s'étendre au cas multi-classe, cette règle peut se réécrire comme

$$\hat{y} = \underset{c=1,\dots,C}{\arg\max}\, \mathbb{P}(Y = c|X = \vec{x}).$$

Nous discuterons plus en détails de cette règle de décision et d'autres dans la section 4.2.

À partir de maintenant, nous écrirons parfois $\mathbb{P}(\vec{x})$ au lieu de $\mathbb{P}(X = \vec{x})$ quand il n'y a pas d'ambiguité.

4.1.2 Loi de Bayes

Le raisonnement probabiliste s'appuie fortement sur la loi de Bayes, qui nous permet d'exprimer la distribution conditionnelle $\mathbb{P}(Y = c|\vec{x})$ comme suit :

Théorème 4.1 (Loi de Bayes)

$$\mathbb{P}(Y = c|\vec{x}) = \frac{\mathbb{P}(Y = c)\mathbb{P}(\vec{x}|Y = c)}{\mathbb{P}(\vec{x})}. \tag{4.2}$$

■

Chacun des éléments de cette équation joue un rôle bien précis et porte ainsi un nom :

- $\mathbb{P}(Y = c)$ est la *distribution a priori* des étiquettes, avant d'avoir observé les données ;
- $\mathbb{P}(\vec{x}|Y = c)$ est la *vraisemblance*. Elle quantifie à quel point il est vraisemblable que l'on observe la réalisation $\vec{x}$ de X sachant que la classe est c ;
- $\mathbb{P}(Y = c|\vec{x})$ est la *distribution a posteriori* des étiquettes, après avoir observé les données.

Le dénominateur $\mathbb{P}(\vec{x})$ est la probabilité marginale que $\vec{x}$ soit observée, indépendamment de sa classe. Il peut être réécrit sous la forme $\mathbb{P}(\vec{x}) = \mathbb{P}(\vec{x}|Y = 0)P(Y = 0) + \mathbb{P}(\vec{x}|Y = 1)P(Y = 1)$. Dans le cas multi-classe, on écrira $\mathbb{P}(\vec{x}) = \sum_{c=1}^{C} \mathbb{P}(\vec{x}|Y = c)P(Y = c)$.

Exemple

Prenons le cas du dépistage du cancer du col de l'utérus par frottis sanguin. Ce test, bien que simple à réaliser, est assez imprécis : il a une sensibilité (la proportion de personnes atteintes pour lequel il est positif) d'environ 70% [1], et une spécificité (la proportion de personnes non atteintes pour lequel il est négatif) d'environ 98%. (Pour plus de détails sur la sensibilité et la spécificité, voir la section 3.2.1.) De plus, l'incidence de la maladie est d'environ 1 femme sur 10 000 [2].

Quelle est la probabilité qu'une personne testée soit atteinte d'un cancer si le test est positif ? C'est un problème d'inférence bayésienne.

Soient X une variable aléatoire à valeurs dans $\{0, 1\}$ modélisant le résultat du test (1 pour positif, 0 pour négatif), et Y une variable aléatoire à valeurs dans $\{0, 1\}$ modélisant le statut de la personne testée (1 pour malade, 0 pour non atteinte).

Inférence : Nous cherchons à calculer $\mathbb{P}(Y = 1|X = 1)$. Appliquons la loi de Bayes :

$$\mathbb{P}(Y = 1|X = 1) = \frac{\mathbb{P}(X = 1|Y = 1)\mathbb{P}(Y = 1)}{\mathbb{P}(X = 1)}.$$

$\mathbb{P}(X = 1|Y = 1)$ n'est autre que la sensibilité du test, autrement dit, $\mathbb{P}(X = 1|Y = 1) = 0,70$. $\mathbb{P}(Y = 1)$ est l'incidence de la maladie, soit 10^{-4}. Enfin, $\mathbb{P}(X = 1) = \mathbb{P}(X = 1|Y = 1)\mathbb{P}(Y = 1) + \mathbb{P}(X = 1|Y = 0)\mathbb{P}(Y = 0)$. Nous connaissons déjà les deux premiers termes de cette équation. De plus, $\mathbb{P}(X = 1|Y = 0) = 1 - 0,90 = 0,02$, et $\mathbb{P}(Y = 0) = 1 - \mathbb{P}(Y = 1) = 0,9999$. Ainsi,

$$\mathbb{P}(Y = 1|X = 1) = \frac{0,70 \times 10^{-4}}{0,70 \times 10^{-4} + 0,02 \times 0,9999} = 0,0035.$$

Prédiction : Ainsi, la probabilité qu'une personne dont le test est positif soit atteinte est seulement de $0,35\%$. Sans autre information, $\mathbb{P}(Y = 0|X = 1) > \mathbb{P}(Y = 0|X = 1)$ et la règle de décision 4.1 retournera une prédiction négative pour tous les tests positifs.

4.1.3 Modélisation paramétrique

Dans l'exemple précédent, $\mathbb{P}(X|Y)$ était donnée. Cependant, il se peut que nous devions aussi modéliser cette distribution. Dans ce cas, nous allons la contraindre à appartenir à une famille bien précise de lois de probabilité, paramétrisée par un vecteur $\vec{\theta}$ à valeurs dans un espace Θ de dimension finie : c'est ce que l'on appelle la *modélisation paramétrique.*

En plus de l'inférence et de la prédiction, nous avons maintenant une tâche supplémentaire : celle de l'*apprentissage*, qui consiste à estimer la valeur du vecteur de paramètres $\vec{\theta}$. Nous verrons comment faire dans la section 4.3.

4.2 Règles de décision

Dans cette section, nous allons voir comment prendre une décision, ou faire une prédiction, à partir des lois de probabilité nécessaires.

1. C'est pourquoi il est recommandé de le faire régulièrement.
2. 6,7 sur 100 000 en 2012

4.2.1 Tests du rapport de vraisemblance

La règle de décision (équation 4.1) qui consiste à prédire la classe la plus probable étant donnée l'observation consiste à sélectionner la classe $\hat{y}$ qui *maximise* la valeur a posteriori $\mathbb{P}(Y = \hat{y}|\vec{x})$. On parle donc de *décision par maximum a posteriori*. En appliquant la loi de Bayes, elle peut être reformulée comme suit :

$$\hat{y} = \begin{cases} 1 & \text{si } \frac{\mathbb{P}(\vec{x}|Y=1)\mathbb{P}(Y=1)}{\mathbb{P}(\vec{x})} > \frac{\mathbb{P}(\vec{x}|Y=0)\mathbb{P}(Y=0)}{\mathbb{P}(\vec{x})} \\ 0 & \text{sinon.} \end{cases}$$

$\mathbb{P}(\vec{x})$ n'affecte pas cette règle de décision et peut donc être éliminée. Remarquons néanmoins que $\mathbb{P}(\vec{x})$ sera nécessaire si nous voulons estimer la valeur a posteriori $\mathbb{P}(Y = \hat{y}|\vec{x})$ pour quantifier la qualité de notre décision.

Définition 4.1 (Décision par maximum a posteriori) Dans le cas binaire, la règle de décision

$$\hat{y} = \begin{cases} 1 & \text{si } \mathbb{P}(\vec{x}|Y = 1)\mathbb{P}(Y = 1) > \mathbb{P}(\vec{x}|Y = 0)\mathbb{P}(Y = 0) \\ 0 & \text{sinon.} \end{cases} \tag{4.3}$$

est appelée *règle de décision par maximum a posteriori*.

Dans le cas multi-classe, cette règle s'écrit

$$\hat{y} = \arg\max_{c=1,\dots,C} \mathbb{P}(\vec{x}|Y = c)\mathbb{P}(Y = c).$$

■

On peut réécrire cette règle en utilisant le rapport des vraisemblances.

Définition 4.2 (Rapport de vraisemblance) On représente par $\Lambda(\vec{x})$ le *rapport de vraisemblance*

$$\Lambda(\vec{x}) = \frac{\mathbb{P}(\vec{x}|Y = 1)}{\mathbb{P}(\vec{x}|Y = 0)},$$

■

La règle de décision par maximum a posteriori peut être reformulée comme ce que l'on appelle un *test du rapport de vraisemblance* :

$$\hat{y} = \begin{cases} 1 & \text{si } \Lambda(\vec{x}) > \frac{\mathbb{P}(Y=0)}{\mathbb{P}(Y=1)} \\ 0 & \text{sinon.} \end{cases} \tag{4.4}$$

Dans le cas où l'on fait l'hypothèse d'égalité des distributions a priori, $\mathbb{P}(Y = 0) = \mathbb{P}(Y = 1)$ et l'on comparera le rapport de vraisemblance à 1.

Définition 4.3 (Décision par maximum de vraisemblance) La règle de décision

$$\hat{y} = \begin{cases} 1 & \text{si } \mathbb{P}(\vec{x}|Y = 1) > \mathbb{P}(\vec{x}|Y = 0) \\ 0 & \text{sinon.} \end{cases}$$

ou, de manière équivalente,

$$\hat{y} = \begin{cases} 1 & \text{si } \Lambda(\vec{x}) > 1 \\ 0 & \text{sinon.} \end{cases}$$

est appelée *règle de décision par maximum de vraisemblance*. ■

Remarque

Dans de nombreux cas, on préfèrera passer au log et évaluer le signe de $\log \Lambda(\vec{x})$.

Exemple

Supposons que nous disposions d'un échantillon d'une population de poissons d'une même espèce, parmi lesquels se trouvent des mâles et des femelles. Nous cherchons à déterminer leur sexe, modélisé comme une variable aléatoire binaire Y (0 pour les mâles, 1 pour les femelles), uniquement à partir de leur longueur, modélisée comme une variable aléatoire continue X. Nous allons supposer la distribution des longueurs des mâles normalement distribuée, centrée en 4 cm et d'écart-type 1 cm, et celle des longueurs des femelles normalement distribuée, centrée en 6 cm, et d'écart-type 1 cm.

$$\mathbb{P}(x|Y=0) \sim \mathcal{N}(4,1) \text{ et } \mathbb{P}(x|Y=1) \sim \mathcal{N}(6,1).$$

Le rapport de vraisemblance s'écrit

$$\frac{\mathbb{P}(x|Y=1)}{\mathbb{P}(x|Y=0)} = \frac{e^{-(x-6)^2/2}}{e^{(x-4)^2/2}}$$

et son logarithme vaut donc

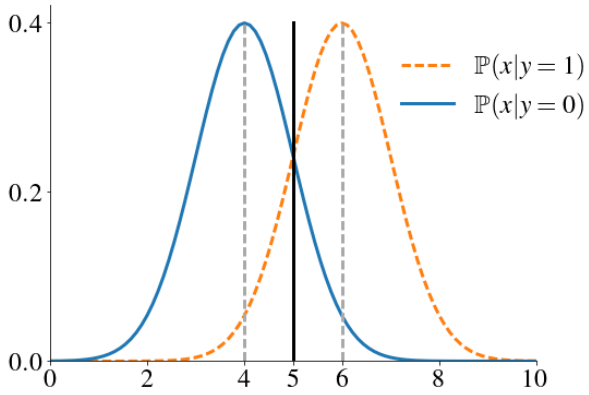
$$\ln \Lambda(x) = -(x-6)^2 + (x-4)^2 = 4(x-5).$$

Ainsi, si l'on suppose les proportions de mâles et de femelles identiques dans notre échantillon, la règle de décision par maximum a posteriori est équivalente à la règle de décision par maximum de vraisemblance et consiste à prédire qu'un poisson est une femelle si sa longueur est supérieure à 5 cm et un mâle sinon. Cette règle de décision est illustrée sur la figure 4.1a.

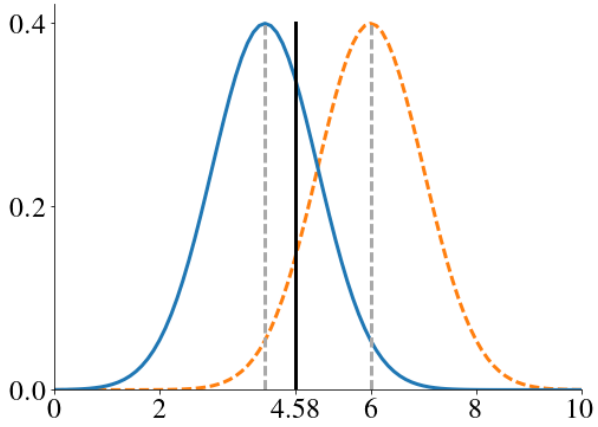
Supposons maintenant que l'on sache avoir cinq fois plus de femelles que de mâles dans notre échantillon. Le rapport des distributions a priori vaut donc $\frac{\mathbb{P}(Y=0)}{\mathbb{P}(Y=1)} = \frac{1}{5}$. Nous comparons donc le logarithme du rapport de vraisemblance à $\ln \frac{1}{5}$. Nous allons prédire qu'un poisson est une femelle si sa longueur est supérieure à $5 - \ln(5)/4 \approx 4.58$: il est plus probable que précédemment qu'un poisson d'une taille légèrement inférieure à 5 cm soit femelle. On observe un déplacement de la valeur seuil en accord avec notre connaissance a priori du problème. Cette règle de décision est illustrée sur la figure 4.1b.

4.2.2 Théorie de la décision bayésienne

Les règles de décision que nous venons de voir s'inscrivent dans le cadre plus général de la *théorie de la décision.* Dans ce cadre, la variable aléatoire Y définie sur $\mathcal{Y}$ représente non pas une étiquette, mais une « vérité » cachée, ou un état de la nature. X est une variable aléatoire définie sur $\mathcal{X}$ qui représente les données observées. Soit de plus une variable A, définie sur un espace, $\mathcal{A}$ qui représente l'ensemble des *décisions* qui peuvent être prises. Nous nous donnons maintenant une *fonction de coût* (*loss function* en anglais), $L : \mathcal{Y} \times \mathcal{A} \to \mathbb{R}$. Cette fonction de coût est à rapprocher de celles que nous avons définies dans la section 2.4. Étant donné une action a et un état caché véritable y, cette fonction quantifie le prix à payer pour avoir choisi l'action a alors que l'état caché véritable était y.



(A) Si les deux classes sont également probables, les poissons dont la taille est inférieure à 5 cm sont étiquetés mâles et les autres femelles.



(B) Si un poisson a cinq fois plus de chances a priori d'être femelle, les poissons dont la taille est inférieure à 4,58 cm sont étiquetés mâles et les autres femelles.

FIGURE 4.1 – Règle de décision pour le sexe d'un guppy en fonction de sa taille.

Exemple

Dois-je prendre ou non mon parapluie ce matin ? Nous pouvons modéliser ce problème de la façon suivante : $\mathcal{A}$ contient deux actions (« prendre mon parapluie » et « ne pas prendre mon parapluie »). $\mathcal{Y}$ contient les vérités « il ne pleut pas », « il pleut un peu », « il pleut fort », « il y a beaucoup de vent ». Enfin, $\mathcal{X}$ est un espace décrivant les informations sur lesquelles je peux m'appuyer, comme les prévisions météorologiques et la couleur du ciel quand je pars de chez moi. Je peux choisir la fonction de coût suivante :

	pas de pluie	pluie faible	pluie forte	vent
parapluie	1	0	0	2
pas de parapluie	0	2	4	0

Dans ce cadre, il est raisonnable de choisir l'action a qui minimise la probabilité d'erreur, autrement dit l'espérance de la fonction de coût :

Définition 4.4 (Décision de Bayes) La règle de décision qui consiste à choisir l'action a^* qui minimise l'espérance de la fonction de coût est appellée *règle de décision de Bayes* :

$$a^*(\vec{x}) = \arg\min_{a \in \mathcal{A}} \mathbb{E}[L(y,a)] = \arg\min_{a \in \mathcal{A}} \sum_{y \in \mathcal{Y}} \mathbb{P}(Y = y|\vec{x}) L(y,a). \tag{4.5}$$

On parlera aussi du principe de *minimisation de la perte espérée* (*minimum expected loss* en anglais.) ■

Remarque

En économie, on préfère au concept de fonction de coût celui d'*utilité*. L'utilité peut être simplement définie comme l'opposé d'une fonction de coût. Le principe ci-dessus s'appelle alors la *maximisation de l'utilité espérée* (*maximum expected utility* en anglais.)

Cette approche est à contraster avec la minimisation du risque empirique 2.3, dans laquelle on remplace la distribution $\mathbb{P}(X, Y)$ par sa distribution *empirique* obtenue en partageant également la masse de probabilité entre les n observations

$$\mathbb{P}(X = \vec{x}, Y = y|\mathcal{D}) = \frac{1}{n}\sum_{i=1}^{n} \delta(y, y^i)\delta(\vec{x}, \vec{x}^{\,i}). \tag{4.6}$$

Par contraste, dans le cadre bayésien, on paramétrise la distribution $\mathbb{P}(X, Y)$ par un paramètre $\vec{\theta}$ optimisé sur $\mathcal{D}$. Dans le cadre empirique, les hypothèses sur la distribution des données sont potentiellement simplistes ; mais dans le cadre bayésien, cette distribution est apprise sans considérer le processus de décision dans lequel elle sera utilisée.

Alors que la décision de Bayes consiste à choisir, pour une observation donnée, l'espérance de la fonction de coût, on définit le risque de Bayes comme l'espérance globale de la fonction de coût :

Définition 4.5 (Risque de Bayes) Le *risque de Bayes* est l'espérance du coût sous la règle de décision de Bayes.

$$r = \int_{\vec{x}\in\mathcal{X}} \sum_{y\in\mathcal{Y}} L(y, a^*(\vec{x}))\mathbb{P}(\vec{x}, y)\,\mathrm{d}\vec{x}. \tag{4.7}$$

■

Définir une stratégie qui minimise le risque de Bayes est équivalent à appliquer la règle de décision de Bayes.

Classification binaire par la règle de décision de Bayes

Revenons au cadre de la classification binaire. $y \in \mathcal{Y}$ représente de nouveau la *véritable* classe d'une observation, tandis que $a \in \mathcal{A}$ représente sa classe *prédite* : $\mathcal{A} = \mathcal{Y} = \{0, 1\}$. La fonction de coût

$$\begin{aligned} L : \mathcal{Y} \times \mathcal{Y} &\to \mathbb{R} \\ c, k &\mapsto \lambda_{ck} \end{aligned}$$

quantifie maintenant le coût de prédire la classe k quand la classe véritable est c (voir section 2.4). La règle de décision de Bayes (équation 4.5) est équivalente à la règle de décision suivante :

$$\hat{y} = \begin{cases} 1 \text{ si } \lambda_{11}\mathbb{P}(Y = 1|\vec{x}) + \lambda_{10}\mathbb{P}(Y = 0|\vec{x}) \leq \lambda_{01}\mathbb{P}(Y = 1|\vec{x}) + \lambda_{00}\mathbb{P}(Y = 0|\vec{x}) \\ 0 \text{ sinon.} \end{cases} \tag{4.8}$$

Elle peut s'écrire sous la forme d'un test du rapport de vraisemblance :

$$\hat{y} = \begin{cases} 1 & \text{si } \frac{\mathbb{P}(\vec{x}|Y=1)}{\mathbb{P}(\vec{x}|Y=0)} > \frac{(\lambda_{01}-\lambda_{00})\mathbb{P}(Y=0)}{(\lambda_{10}-\lambda_{11})\mathbb{P}(Y=1)} \\ 0 & \text{sinon.} \end{cases} \tag{4.9}$$

Dans le cas multi-classe, cette règle devient :

$$\hat{y} = \underset{c=1,\ldots,C}{\arg\min} \sum_{k=1}^{K} \lambda_{kc}\mathbb{P}(Y = k|\vec{x}).$$

Coût 0/1

On retrouve le coût 0/1 (section 2.4) en utilisant $\lambda_{ck} = 1 - \delta(k, c)$. L'équation 4.8 devient

$$\hat{y} = \begin{cases} 1 & \text{si } \mathbb{P}(Y = 0|\vec{x}) \leq \mathbb{P}(Y = 1|\vec{x}) \\ 0 & \text{sinon} \end{cases} \tag{4.10}$$

et ainsi la règle de décision de Bayes est équivalente à la règle décision par maximum a posteriori. Ceci est vrai aussi dans le cas multi-classe.

Le coût 0/1 n'est pas la seule fonction de coût possible, même pour un problème de classification binaire. En particulier, toutes les erreurs de classification ne sont pas nécessairement également coûteuses. Par exemple, prédire qu'une patiente atteinte d'un cancer est en bonne santé peut être largement plus problématique que l'inverse.

Régions de décision

Les règles de décisions peuvent aussi s'exprimer en termes de régions de décision (cf. section 2.1) : la règle de décision consiste simplement à étiqueter l'observation $\vec{x}$ en accord avec la région de décision à laquelle elle appartient :

$$\hat{y} = \begin{cases} 1 & \text{si } \vec{x} \in \mathcal{R}_1 \\ 0 & \text{sinon.} \end{cases} \tag{4.11}$$

Dans le cas multi-classe, cette règle revient à

$$\hat{y} = \sum_{c=1}^{C} \delta_{\vec{x} \in \mathcal{R}_c}.$$

Cette règle de décision est équivalente à la règle de décision de Bayes si l'on définit comme fonction discriminante la fonction :

$$\begin{aligned} f(\vec{x}) = &(\lambda_{01}\mathbb{P}(Y = 1|\vec{x}) + \lambda_{00}\mathbb{P}(Y = 0|\vec{x})) - \\ &(\lambda_{11}\mathbb{P}(Y = 1|\vec{x}) + \lambda_{10}\mathbb{P}(Y = 0|\vec{x}))\,, \end{aligned} \tag{4.12}$$

ou, dans le cas multi-classe

$$f_c(\vec{x}) = -\sum_{k=1}^{C} \lambda_{ck}\mathbb{P}(Y = k|\vec{x}).$$

Dans le cas du coût 0/1, la fonction discriminante vaut $f(\vec{x}) = \mathbb{P}(Y = 1|\vec{x}) - \mathbb{P}(Y = 0|\vec{x})$ et la règle de décision de Bayes est bien équivalente à la décision par maximum a posteriori (équation 4.4).

Le risque de Bayes(4.7) peut s'exprimer en fonction des régions de décision :

$$\begin{aligned} r &= \int_{\vec{x} \in \mathcal{X}} \lambda_{0,a^*(\vec{x})}\mathbb{P}(\vec{x}|Y = 0)\mathbb{P}(Y = 0) + \lambda_{1,a^*(\vec{x})}\mathbb{P}(\vec{x}|Y = 1)\mathbb{P}(Y = 1)\,\mathrm{d}\vec{x} \\ &= \int_{\vec{x} \in \mathcal{R}_0} \lambda_{00}\mathbb{P}(\vec{x}|Y = 0)\mathbb{P}(Y = 0) + \lambda_{10}\mathbb{P}(\vec{x}|Y = 1)\mathbb{P}(Y = 1)\,\mathrm{d}\vec{x} \\ &\quad + \int_{\vec{x} \in \mathcal{R}_1} \lambda_{01}\mathbb{P}(\vec{x}|Y = 0)\mathbb{P}(Y = 0) + \lambda_{11}\mathbb{P}(\vec{x}|Y = 1)\mathbb{P}(Y = 1)\,\mathrm{d}\vec{x}. \end{aligned} \tag{4.13}$$

Définir les régions de décision de sorte à minimiser le risque de Bayes est équivalent à utiliser la fonction discriminante définie par l'équation 4.12. Cela vaut aussi pour le cas multi-classe, où

$$r = \sum_{k=1}^{C} \int_{\vec{x} \in \mathcal{R}_k} \sum_{c=1}^{C} \lambda_{ck}\mathbb{P}(\vec{x}|Y = c)\mathbb{P}(Y = c)\,\mathrm{d}\vec{x}.$$

Rejet

Pour certaines applications, il peut être intéressant que l'algorithme refuse de prendre une décision quand sa confiance en sa prédiction est faible, autrement dit, quand le rapport de vraisemblance est très proche de 1. Dans ce cas, on pourra rajouter une classe artificielle $C+1$ de *rejet*. On peut adapter le coût 0/1 à ce cas en proposant

$$\lambda_{ck} = \begin{cases} 0 & \text{si } k = c \\ \lambda & \text{si } k = C+1 \\ 1 & \text{sinon,} \end{cases} \tag{4.14}$$

avec $0 < \lambda < 1$. La règle de décision par minimisation de la perte espérée devient alors

$$\hat{y} = \begin{cases} c & \text{si } \mathbb{P}(Y = c|\vec{x}) \geq \mathbb{P}(Y = k|\vec{x}) \quad \forall k \neq c \\ \text{rejet} & \text{sinon.} \end{cases} \tag{4.15}$$

Dans ce cas, les régions de décision $\mathcal{R}_1, \ldots \mathcal{R}_C$ ne couvrent pas $\mathcal{X}$ et la réunion de leur complémentaire est la zone de rejet.

4.3 Estimation de densité

Supposons disposer d'un échantillon $\mathcal{D} = x^1, x^2, \ldots, x^n$ constitué n observations d'une variable aléatoire X, à valeurs sur $\mathcal{X}$. Nous allons de plus supposer que la distribution de X a une forme connue et est paramétrisée par le paramètre θ. Comment *estimer* θ ?

4.3.1 Estimation par maximum de vraisemblance

Définition 4.6 (Estimateur par maximum de vraisemblance) L'estimateur par *maximum de vraisemblance* (*maximum likelihood estimator* ou *MLE* en anglais) de θ est le vecteur $\hat{\theta}_{\text{MLE}}$ qui maximise la vraisemblance, autrement dit la probabilité d'observer $\mathcal{D}$ étant donné θ :

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta} \mathbb{P}(\mathcal{D}|\theta). \tag{4.16}$$

■

Pour trouver $\hat{\theta}_{\text{MLE}}$, nous allons supposer que les n observations sont *indépendantes et identiquement distribuées* (ou *iid*). Cela nous permet de décomposer la vraisemblance comme

$$\mathbb{P}(\mathcal{D}|\theta) = \prod_{i=1}^{n} \mathbb{P}(X = \vec{x}^{\,i}|\theta). \tag{4.17}$$

Enfin, remarquons que pour simplifier les calculs, on choisira souvent de maximiser non pas directement la vraisemblance mais son logarithme :

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta} \sum_{i=1}^{n} \log \mathbb{P}(X = \vec{x}^{\,i}|\theta). \tag{4.18}$$

Exemple

Prenons l'exemple d'un jeu de pile ou face. Nous modélisons l'observation « pile » ou « face » comme la réalisation d'une variable aléatoire X, définie sur l'univers $\mathcal{X} = \{0, 1\}$ (0 correspondant à « pile » et 1

à « face »), et suivant une loi de probabilité $\mathbb{P}$. Un choix classique pour cette loi de probabilité est d'utiliser une *loi de Bernoulli* :

$$\mathbb{P}(X = x) = \begin{cases} p & \text{si } x = 1 \\ (1-p) & \text{si } x = 0, \end{cases} \tag{4.19}$$

ou, de manière équivalente, $\mathbb{P}(X = x) = p^x(1-p)^{1-x}$. Supposons $\mathcal{D} = \{x^1, x^2, \ldots, x^n\}$ constitué de n observations iid. D'après l'équation 4.18, l'estimateur par maximum de vraisemblance de p est

$$\begin{aligned} \hat{p}_{\text{MLE}} &= \arg\max_{p\in[0,1]} \sum_{i=1}^{n} \log \mathbb{P}(X = x^i|p) = \arg\max_{p\in[0,1]} \sum_{i=1}^{n} \log \left(p^{x^i}(1-p)^{1-x^i}\right) \\ &= \arg\max_{p\in[0,1]} \sum_{i=1}^{n} x^i \log p + \left(n - \sum_{i=1}^{n} x^i\right) \log(1-p). \end{aligned}$$

La fonction $L : p \mapsto \sum_{i=1}^{n} x^i \log p + \left(n - \sum_{i=1}^{n} x^i\right) \log(1-p)$ est concave, nous pouvons donc la maximiser en annulant sa dérivée :

$$\frac{\partial L}{\partial p} = \sum_{i=1}^{n} x^i \frac{1}{p} - \left(n - \sum_{i=1}^{n} x^i\right) \frac{1}{1-p},$$

ce qui nous donne

$$(1 - \hat{p}_{\text{MLE}}) \sum_{i=1}^{n} x^i - \hat{p}_{\text{MLE}} \left(n - \sum_{i=1}^{n} x^i\right) = 0$$

et donc

$$\hat{p}_{\text{MLE}} = \frac{1}{n} \sum_{i=1}^{n} x^i. \tag{4.20}$$

L'estimateur par maximum de vraisemblance de p est tout simplement la moyenne de l'échantillon.

Exemple

Reprenons l'exemple du test de dépistage du cancer du col de l'utérus. Alors que dans l'exemple précédent, $\mathbb{P}(X|Y = 0)$ et $\mathbb{P}(X|Y = 1)$ étaient données, nous allons maintenant les estimer à partir de données observées. Supposons que nous observions un jeu $\mathcal{D}_0$ de n_0 personnes non atteintes, parmi lesquelles t_0 ont un test négatif, et un jeu $\mathcal{D}_1$ de n_1 personnes atteintes, parmi lesquelles t_1 ont un test positif. La prévalence de la maladie est $\mathbb{P}(Y = 1) = \pi$. Nous pouvons modéliser la probabilité $\mathbb{P}(X|Y = 1)$ par une loi de Bernoulli de paramètre p_1, et $\mathbb{P}(X|Y = 0)$ par une loi de Bernoulli de paramètre p_0.

La probabilité qu'une personne dont le test est positif soit atteinte est

$$\mathbb{P}(Y = 1|X = 1) = \frac{\mathbb{P}(X = 1|Y = 1)\mathbb{P}(Y = 1)}{\mathbb{P}(X = 1)}.$$

Nous savons que $\mathbb{P}(X = x|Y = 0) = p_0^x(1-p_0)^{1-x}$ et $\mathbb{P}(X = x|Y = 1) = p_1^x(1-p_1)^{1-x}$. Ainsi,

$$\mathbb{P}(Y = 1|X = 1) = \frac{p_1\pi}{p_1\pi + p_0(1-\pi)}.$$

Nous pouvons remplacer dans cette équation p_0 et p_1 par leurs estimateurs par maximum de vraisemblance (équation 4.20) :

$$\hat{p}_0 = 1 - \frac{t_0}{n_0} \text{ et } \hat{p}_1 = \frac{t_1}{n_1}. \tag{4.21}$$

Il s'agit respectivement de la spécificité et de la sensibilité estimées du test. En utilisant $\frac{t_0}{n_0} = 0,98$, $\frac{t_1}{n_1} = 0,70$ et $\pi = 10^{-5}$, on retrouve les résultats précédents.

4.3.2 Estimateur de Bayes

Supposons que plutôt que de ne pas connaître du tout la valeur du paramètre θ, nous ayons, en tant qu'expert du domaine d'application, une bonne idée des valeurs qu'il peut prendre. Cette information peut être très utile, surtout quand le nombre d'observations est faible. Pour l'utiliser, nous allons modéliser θ à son tour comme une variable aléatoire, et définir sa distribution a priori $\mathbb{P}(\theta)$.

Définition 4.7 (Estimateur de Bayes) Étant donnée une fonction de coût L, l'*estimateur de Bayes* $\hat{\theta}_{\text{Bayes}}$ de θ est défini par

$$\hat{\theta}_{\text{Bayes}} = \arg\min_{\hat{\theta}} \mathbb{E}[L(\theta, \hat{\theta})]. \tag{4.22}$$

■

Si l'on utilise pour L l'erreur quadratique moyenne, alors

$$\hat{\theta}_{\text{Bayes}} = \arg\min_{\hat{\theta}} \mathbb{E}[(\theta - \hat{\theta})^2] \tag{4.23}$$

En considérant $\hat{\theta}$ déterministe, nous avons

$$\begin{aligned}
\hat{\theta}_{\text{Bayes}} &= \arg\min_{\hat{\theta}} \hat{\theta}^2 - 2\hat{\theta}\mathbb{E}[\theta] + \mathbb{E}[\theta^2] \\
&= \arg\min_{\hat{\theta}} \Big(\hat{\theta} - \mathbb{E}[\theta]\Big)^2 - \mathbb{E}[\theta]^2 + \mathbb{E}[\theta^2] \\
&= \mathbb{E}[\theta].
\end{aligned}$$

Cette dernière égalité s'obtient en remarquant que ni $\mathbb{E}[\theta]$ ni $\mathbb{E}[\theta^2]$ ne dépendent de $\hat{\theta}$. Cette espérance est prise sur la distribution de θ et de X, qui nous sert à estimer θ ; ainsi

$$\hat{\theta}_{\text{Bayes}} = \mathbb{E}[\theta|X] = \int \theta \mathbb{P}(\theta|X)\, \mathrm{d}\theta. \tag{4.24}$$

Quand la distribution a priori du paramètre est uniforme, l'estimateur de Bayes est équivalent à l'estimateur par maximum de vraisemblance.

Exemple

Reprenons notre exemple de dépistage du cancer du col de l'utérus, et supposons maintenant que p_0 et p_1 suivent chacun une loi bêta de paramètres (α_0, β_0) et (α_1, β_1) respectivement. La densité de probabilité de la *loi bêta* de paramètres $\alpha, \beta > 0$, définie sur $0 \leq u \leq 1$, est donnée par :

$$f_{\alpha,\beta}(u) = \frac{u^{\alpha-1}(1-u)^{\beta-1}}{B(\alpha, \beta)} \tag{4.25}$$

où $B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$ et Γ est la fonction gamma. L'espérance de cette loi est $\frac{\alpha}{\alpha+\beta}$.

Pour calculer l'estimateur de Bayes de p_0, il nous faut connaître la loi $\mathbb{P}(p_0|\mathcal{D}_0)$.

La loi de Bayes nous permet d'écrire

$$\begin{aligned}
\mathbb{P}(p_0|\mathcal{D}_0) &= \frac{\mathbb{P}(\mathcal{D}_0|p_0)\mathbb{P}(p_0)}{\mathbb{P}(\mathcal{D}_0)} \\
&= \frac{1}{\mathbb{P}(\mathcal{D}_0)B(\alpha_0, \beta_0)} \prod_{i=1}^{n_0} p_0^{x^i}(1-p_0)^{1-x^i} p_0^{\alpha_0-1}(1-p_0)^{\beta_0-1} \quad \text{(hypothèse iid)} \\
&= \frac{1}{\mathbb{P}(\mathcal{D}_0)B(\alpha_0, \beta_0)} p_0^{n_0-t_0+\alpha_0-1}(1-p_0)^{t_0+\beta_0-1}.
\end{aligned}$$

Ainsi $p_0|\mathcal{D}_0$ suit une distribution beta de paramètres $(n_0 - t_0 + \alpha_0)$ et $(t_0 + \beta_0)$. L'estimateur de Bayes de p_0 est ainsi

$$\widetilde{p}_0 = \mathbb{E}[p_0|\mathcal{D}_0] = \frac{(n_0 - t_0 + \alpha_0)}{(n_0 - t_0 + \alpha_0) + (t_0 + \beta_0)} = \frac{n_0 - t_0 + \alpha_0}{n_0 + \alpha_0 + \beta_0}. \tag{4.26}$$

En utilisant l'équation 4.21, on peut réécrire l'équation 4.26 sous la forme

$$\widetilde{p}_0 = \frac{n_0}{n_0 + \alpha_0 + \beta_0}\widehat{p}_0 + \frac{\alpha_0 + \beta_0}{n_0 + \alpha_0 + \beta_0}\frac{\alpha_0}{\alpha_0 + \beta_0}.$$

Quand le nombre d'observations n_0 est grand, l'estimateur de Bayes $\widetilde{p}_0$ est proche de l'estimateur par maximum de vraisemblance $\widehat{p}_0$. À l'inverse, si n_0 est faible, l'estimateur de Bayes est proche de $\frac{\alpha_0}{\alpha_0+\beta_0}$ qui est l'espérance de la distribution a priori sur p_0. Ainsi, plus on a de données, plus on leur fait confiance et plus l'estimateur s'éloigne de l'espérance a priori du paramètre, dont on sera plus proche avec peu d'observations.

Le même raisonnement s'applique à p_1, dont l'estimateur de Bayes est

$$\widetilde{p}_1 = \frac{t_1 + \alpha_1}{n_1 + \alpha_1 + \beta_1} = \frac{n_1}{n_1 + \alpha_1 + \beta_1}\widehat{p}_1 + \frac{\alpha_1 + \beta_1}{n_1 + \alpha_1 + \beta_1}\frac{\alpha_1}{\alpha_1 + \beta_1}. \tag{4.27}$$

4.3.3 Décomposition biais-variance

Pour un paramètre θ, l'erreur quadratique moyenne d'un estimateur $\hat{\theta}$ est

$$\begin{aligned}
\text{MSE}(\hat{\theta}) &= \mathbb{E}[(\hat{\theta} - \theta)^2] \\
&= \mathbb{E}[(\hat{\theta} - \mathbb{E}[\hat{\theta}] + \mathbb{E}[\hat{\theta}] - \theta)^2] \\
&= \mathbb{E}[(\hat{\theta} - \mathbb{E}[\hat{\theta}])^2] + \mathbb{E}[(\mathbb{E}[\hat{\theta}] - \theta)^2] + \mathbb{E}[2(\hat{\theta} - \mathbb{E}[\hat{\theta}])(\mathbb{E}[\hat{\theta}] - \theta)] \\
&= \text{Var}(\hat{\theta}) + (\mathbb{E}[\hat{\theta}] - \theta)^2.
\end{aligned}$$

Cette dernière égalité est obtenue en remarquant que $\mathbb{E}[\hat{\theta}] - \theta$ est déterministe et que $\mathbb{E}[(\mathbb{E}[\hat{\theta}] - \theta)] = \mathbb{E}[\mathbb{E}[\hat{\theta}]] - \mathbb{E}[\theta] = 0$. Ainsi l'erreur quadratique moyenne d'un estimateur est la somme de sa variance et du carré de son biais. C'est pourquoi un estimateur biaisé peut, si sa variance est plus faible, avoir une erreur quadratique moyenne plus faible qu'un estimateur non biaisé. On retrouve ici la notion de compromis biais-variance de la section 2.5.3.

4.4 Classification naïve bayésienne

Nous allons maintenant mettre en œuvre les principes que nous venons de voir pour élaborer notre premier algorithme d'apprentissage automatique.

4.4.1 Principes

Dans les exemples que nous avons vu jusqu'à présent, les variables aléatoires que nous avons utilisées étaient unidimensionnelles. Cependant, dans la plupart des applications, les données sont multi-dimensionnelles. Cela peut grandement compliquer les calculs, sauf dans le cas où nous faisons l'hypothèse *naïve* que les variables sont conditionnellement indépendantes.

Supposons n observations en p dimensions : $\{\vec{x}^{\,1}, \vec{x}^{\,2}, \ldots, \vec{x}^{\,n}\}$ telles que $\vec{x}^{\,i} \in \mathbb{R}^p$, et leurs étiquettes $y^1, y^2, \ldots, y^n$. Nous allons modéliser chacune des p variables comme une variable aléatoire X_j. L'hypothèse d'indépendance conditionnelle signifie que

$$\mathbb{P}(X_j = x_j | Y = y, X_m = x_m) = \mathbb{P}(X_j = x_j | Y = y) \qquad \forall 1 \leq j \neq m \leq p. \tag{4.28}$$

Cette hypothèse permet d'écrire la vraisemblance de la façon suivante :

$$\mathbb{P}(Y = y | X_1 = x_1, X_2 = x_2, \ldots, X_p = x_p) = \frac{\prod_{j=1}^{p} \mathbb{P}(X_j = x_j | Y = y) \mathbb{P}(Y = y)}{P(X_1 = x_1, X_2 = x_2, \ldots, X_p = x_p)} \tag{4.29}$$

Définition 4.8 (Classifieur bayésien naïf) On appelle *classifieur bayésien naïf*, ou *naive Bayes classifier* en anglais, un classifieur construit en appliquant la règle de décision par maximum a posteriori à des observations multi-dimensionnelles dont on suppose que les variables sont indépendantes conditionnellement à l'étiquette.

La règle de décision prend la forme

$$\widehat{y} = \underset{c=1,\ldots,C}{\arg\max}\, \mathbb{P}(Y = c) \prod_{j=1}^{p} \mathbb{P}(X_j = x_j | Y = c). \tag{4.30}$$

■

4.4.2 Filtrage bayésien du spam

Un cas classique d'application de la classification naïve bayésienne est celui du filtrage de courier électronique pour séparer les spams des e-mails légitimes. Nous allons supposer disposer de n e-mails et de leurs étiquettes binaires, $y^i = 1$ s'il s'agit d'un spam et $y^i = 0$ sinon.

Représentation d'un e-mail par p variables

Notre première tâche est de trouver une représentation p-dimensionnelle d'e-mails. Pour ce faire, nous allons choisir p mots clés, tels que « riche », « célibataire », « gagner », « cliquer », dont la présence sera informative sur la probabilité du message d'être un spam. En pratique, on peut utiliser l'ensemble des mots qui apparaissent dans les e-mails de notre base de données d'entraînement, en excluant éventuellement les mots neutres tels que les articles (« le », « la », « un »).

Un e-mail sera ensuite représenté par p variables binaires, chacune correspondant à un des mots de notre liste, et valant 1 si le mot apparaît dans l'e-mail, 0 sinon.

Nous allons donc modéliser chacune de ces variables comme la réalisation d'une variable aléatoire binaire X_j suivant une loi de Bernoulli de paramètre p_j. Nous modélisons aussi l'étiquette comme la réalisation d'une variable aléatoire binaire Y.

Pour pouvoir utiliser un classifieur naïf bayésien, nous allons supposer que les variables sont indépendantes conditionnellement à l'étiquette. Cette hypothèse est effectivement naïve : parmi les spams, certains mots sont plus probables une fois qu'on sait qu'un autre mot est présent (par exemple, « célibataire » est plus susceptible d'apparaître une fois que l'on sait que le mot « rencontre » est présent.) En pratique, cela n'empêche pas le classifieur d'avoir d'assez bons résultats.

Maximum a posteriori

La règle de décision par maximum a posteriori (équation 4.30) est

$$\widehat{y} = \begin{cases} 1 & \text{si } \mathbb{P}(Y=1)\prod_{j=1}^{p}\mathbb{P}(X_j = x_j|Y=1) > \mathbb{P}(Y=0)\prod_{j=1}^{p}\mathbb{P}(X_j = x_j|Y=0) \\ 0 & \text{sinon.} \end{cases} \tag{4.31}$$

Inférence

Il nous reste donc à déterminer $\mathbb{P}(Y=1)$, $\mathbb{P}(Y=0)$, et pour chaque variable X_j, $\mathbb{P}(X_j = x_j|Y=0)$ et $\mathbb{P}(X_j = x_j|Y=1)$.

$\mathbb{P}(Y=1)$ est tout simplement la fréquence des spams dans notre jeu de données d'entraînement, et $\mathbb{P}(Y=0) = 1 - \mathbb{P}(Y=1)$ la fréquence des e-mails légitimes. On peut aussi utiliser des statistiques sur la question, qui donnent la proportion de spams parmi les e-mails qui circulent aux alentours de 80%.

Comme nous avons fait le choix de modéliser X_j comme une variable aléatoire suivant une loi de Bernoulli, $\mathbb{P}(X_j = x_j|Y=1) = p_j^{x_j}(1-p_j)^{1-x_j}$. On pourrait utiliser, pour estimer p_j, l'estimateur par maximum de vraisemblance (équation 4.20) : $\widehat{p}_j = \frac{S_j}{S}$ où S_j est le nombre de spams contenant le j-ème mot et S le nombre de spams dans notre jeu de données.

Cependant, cet estimateur est peu adapté aux mots rares : si notre j-ème mot n'apparaît pas du tout dans le jeu d'entraînement, $S_j = S = 0$. Pour cette raison, on préfèrera appliquer ce que l'on appelle un *lissage de Laplace* pour obtenir l'estimateur suivant :

$$\widehat{p}_j = \frac{S_j + 1}{S + 2}. \tag{4.32}$$

Pour un mot qui n'apparaît pas dans le jeu d'entraînement, $\widehat{p}_j = 0,5$.

4.5 Sélection de modèle bayésienne

Le cadre bayésien donne un nouvel éclairage sur la notion de sélection de modèle (voir chapitre 3). En effet, on peut appliquer la règle de Bayes à la probabilité jointe d'un modèle $\mathcal{M}$ et d'un jeu de données $\mathcal{D}$:

$$\mathbb{P}(\mathcal{M}|\mathcal{D}) = \frac{\mathbb{P}(\mathcal{D}|\mathcal{M})\mathbb{P}(\mathcal{M})}{\mathbb{P}(\mathcal{D})}. \tag{4.33}$$

En prenant le logarithme, on peut écrire de manière équivalente

$$\log\mathbb{P}(\mathcal{M}|\mathcal{D}) = \log\mathbb{P}(\mathcal{D}|\mathcal{M}) + \log\mathbb{P}(\mathcal{M}) - \log\mathbb{P}(\mathcal{D}). \tag{4.34}$$

La maximisation a posteriori de l'équation 4.34 est équivalente à minimiser la somme d'un terme caractérisant l'erreur empirique ($-\log\mathbb{P}(\mathcal{D}|\mathcal{M})$) et d'un terme caractérisant la complexité du modèle ($-\log\mathbb{P}(\mathcal{M})$). Cette formulation est similaire à celle de la *régularisation* (voir section 2.5.4). La régularisation peut alors être comprise comme le fait de choisir une distribution a priori pour $\mathcal{M}$ qui favorise les modèles moins complexes.

C'est dans ce cadre que s'inscrit par exemple le critère de sélection bayésien (BIC). Le détailler dépasse l'ambition de cet ouvrage.

Points clefs

— La modélisation d'un problème d'apprentissage de manière probabiliste se divise en deux tâches :
  — Un problème d'inférence, qui consiste à déterminer la loi de probabilité $\mathbb{P}(Y = c|X)$;
  — Une tâche de prédiction, qui consiste à appliquer une règle de décision.

— Le raisonnement probabiliste s'appuie sur la loi de Bayes

— Généralement, la loi de probabilité $\mathbb{P}(Y = c|X)$ est modélisée de manière paramétrique et le problème d'inférence se résout en s'appuyant d'une part sur la loi de Bayes, et d'autre part sur des estimateurs tels que l'estimateur par maximum de vraisemblance ou l'estimateur de Bayes pour en déterminer les paramètres.

— La règle de décision de Bayes, qui consiste à minimiser l'*espérance* de la perte, contraste avec la règle de minimisation du risque *empirique.*

Pour aller plus loin

- Nous avons uniquement abordé dans ce chapitre des problèmes de classification. Cependant, la stratégie de minimisation du risque de Bayes s'applique au cas de la régression, en utilisant une fonction de perte appropriée. Dans ce cadre, les sommations sur y seront remplacées par des intégrales.
- L'ouvrage de Ross (1987) détaille l'estimation par maximum de vraisemblance et l'estimation bayésienne.
- Le parti pris dans cet ouvrage est de ne pas aborder en plus de détails l'approche bayésienne du machine learning. Les curieux pourront se pencher sur les livres de Murphy (2013) et Barber (2012).
- L'article de Hand et Yu (2001) analyse en détails la classification naïve bayésienne.

Bibliographie

Barber, D. (2012). *Bayesian Reasoning and Machine Learning*. Cambridge University Press. `http://www.cs.ucl.ac.uk/staff/d.barber/brml/`.

Hand, D. J. et Yu, K. (2001). Idiot's Bayes : not so stupid after all ? *International Statistical Review / Revue Internationale de Statistique*, 69(3) :385–398.

Murphy, K. P. (2013). *Machine Learning : A Probabilistic Perspective*. MIT Press, Cambridge, MA, 4th edition. `https://www.cs.ubc.ca/~murphyk/MLbook/`.

Ross, S. M. (1987). *Introduction to Probability and Statistics for Engineers and Scientists*. Wiley, New York.

Chapitre 5

Régressions paramétriques

Un modèle de régression paramétrique suppose que la forme analytique de la fonction de décision est connue. Dans ce contexte, ce chapitre se concentre principalement sur des problèmes de *régression linéaire*, c'est-à-dire ceux pour lesquels la fonction de décision est une fonction linéaire des descripteurs.

Les modèles linéaires ont une longue histoire dans le domaine des statistiques, depuis bien avant le développement des ordinateurs ; mais nous ne les étudions pas que pour cette raison. En effet, malgré leur simplicité, ils peuvent avoir de bonnes performances, meilleures parfois que celles de modèles non linéaires plus populaires (surtout dans le cas où la taille du jeu d'entraînement est faible). De plus, ces modèles sont facilement interprétables. Enfin, leur compréhension est une excellente base sur laquelle construire des modèles non linéaires.

Ce chapitre présente en détail la régression linéaire et son application aux problèmes de classification avec la régression logistique. Il se termine par quelques mots sur la régression polynomiale.

Objectifs

— Distinguer un modèle d'apprentissage paramétrique d'un modèle non paramétrique

— Formuler un problème de régression linéaire ou logistique

— Résoudre un problème de régression paramétrique

5.1 Apprentissage supervisé d'un modèle paramétrique

5.1.1 Modèles paramétriques

On parle de *modèle paramétrique* quand on utilise un algorithme d'apprentissage dont le but est de trouver les valeurs optimales des paramètres d'un modèle dont on a défini la forme analytique en fonction des descripteurs (cf. section 4.1.3).

La complexité d'un modèle paramétrique grandit avec le nombre de paramètres à apprendre, autrement dit avec le nombre de variables. À l'inverse, la complexité d'un modèle non paramétrique aura tendance à grandir avec le nombre d'observations.

Exemple

Un algorithme d'apprentissage qui permet d'apprendre les coefficient α, β, γ dans la fonction de décision suivante : $f : \vec{x} \mapsto \alpha x_1 + \beta x_2 x_4^2 + \gamma e^{x_3 - x_5}$ apprend un modèle paramétrique. Quel que soit le nombre d'observations, ce modèle ne change pas.

À l'inverse, la méthode du plus proche voisin, qui associe à $\vec{x}$ l'étiquette du point du jeu d'entraînement dont il est le plus proche en distance euclidienne, apprend un modèle non paramétrique : on ne sait pas écrire la fonction de décision comme une fonction des variables prédictives. Plus il y a d'observations, plus le modèle pourra apprendre une frontière de décision complexe (cf. chapitre 8).

Étant donné un jeu $\mathcal{D} = \{\vec{x}^{\,i}, y^i\}_{i=1,\dots,n}$ de n observations en p dimensions et leurs étiquettes réelles, nous supposons ici que la fonction de décision f est paramétrée par le vecteur $\vec{\beta} \in \mathbb{R}^m$.

Nous allons faire l'hypothèse que les erreurs, c'est-à-dire la différence entre les étiquettes réelles et les valeurs correspondantes de f, sont normalement distribuées, centrées en 0 :

$$y = f(\vec{x}|\vec{\beta}) + \epsilon \qquad \epsilon \sim \mathcal{N}(0, \sigma^2). \tag{5.1}$$

Cette hypothèse est illustrée sur la figure 5.1 dans le cas d'une fonction de décision linéaire.

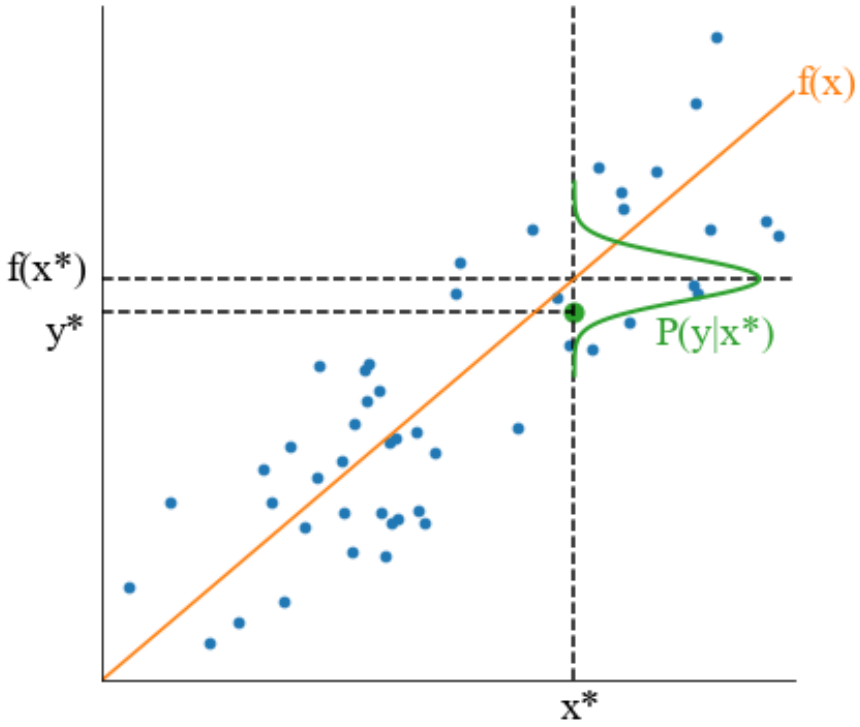


FIGURE 5.1 – Pour une observation x^* donnée (ici en une dimension) , la distribution des valeurs possibles de l'étiquette y^* correspondante est une gaussienne centrée en $f(x^*)$.

Selon cette hypothèse, les observations $\vec{x}$ sont la réalisation de p variables aléatoires $X_1, X_2, \dots, X_p$ à valeurs réelles, leurs étiquettes y sont la réalisation d'une variable aléatoire Y à valeurs réelles, et ces variables aléatoires vérifient

$$\mathbb{P}(Y = y|X = \vec{x}) \sim \mathcal{N}\left(f(\vec{x}|\vec{\beta}), \sigma^2\right). \tag{5.2}$$

On note ici $\mathbb{P}(X = \vec{x})$ pour $\mathbb{P}(X_1 = x_1, X_2 = x_2, \dots, X_p = x_p)$.

5.1.2 Estimation par maximum de vraisemblance et méthode des moindres carrés

Sous l'hypothèse 5.2, et en supposant les n observations indépendantes et identiquement distribuées, le log de vraisemblance du paramètre $\vec{\beta}$ vaut :

$$\begin{aligned}\log \mathbb{P}(\mathcal{D}|\vec{\beta}) &= \log \prod_{i=1}^{n} \mathbb{P}(X = \vec{x}^{\,i}|\vec{\beta}) \\ &= \log \prod_{i=1}^{n} \mathbb{P}(y^i|\vec{x}^{\,i}) + \log \prod_{i=1}^{n} \mathbb{P}(X = \vec{x}^{\,i}) \\ &= -\log \frac{1}{2\sigma^2} \sum_{i=1}^{n} \left(y^i - f(\vec{x}^{\,i}|\vec{\beta})\right)^2 + \mathcal{C}\end{aligned}$$

Dans cette dernière équation, $\mathcal{C}$ est une constante par rapport à $\vec{\beta}$, et provient d'une part du coefficient $\frac{1}{\sqrt{2\pi}}$ de la distribution normale et d'autre part des $\mathbb{P}(X = \vec{x}^{\,i})$.

Ainsi, maximiser la vraisemblance revient à minimiser $\sum_{i=1}^{n} \left(y^i - f(\vec{x}^{\,i}|\vec{\beta})\right)^2$: c'est ce que l'on appelle la *minimisation des moindres carrés*, une méthode bien connue depuis Gauss et Legendre. Notons aussi que cela revient à minimiser le risque empirique quand il est défini en utilisant la fonction de coût quadratique 2.4.3.

5.2 Régression linéaire

Commençons par considérer des modèles *linéaires* : nous cherchons à expliquer la variable cible y par une *combinaison linéaire* – en d'autres mots une somme pondérée – des descripteurs.

5.2.1 Formulation

Nous choisissons une fonction de décision f de la forme

$$f : \vec{x} \mapsto \beta_0 + \sum_{j=1}^{p} \beta_j x_j. \tag{5.3}$$

Ici, $\vec{\beta} \in \mathbb{R}^{p+1}$ et donc $m = p + 1$.

5.2.2 Solution

Définition 5.1 (Régression linéaire) On appelle *régression linéaire* le modèle de la forme $f : \vec{x} \mapsto \beta_0 + \sum_{j=1}^{p} \beta_j x_j$ dont les coefficients sont obtenus par minimisation de la somme des moindres carrés, à savoir :

$$\arg\min_{\vec{\beta} \in \mathbb{R}^{p+1}} \sum_{i=1}^{n} \left(y^i - \left(\beta_0 + \sum_{j=1}^{p} \beta_j x_j\right)\right)^2. \tag{5.4}$$

■

Nous pouvons réécrire le problème 5.4 sous forme matricielle, en ajoutant à gauche à la matrice d'observations $X \in \mathbb{R}^p$ une colonne de 1 :

$$X \leftarrow \begin{pmatrix} 1 & x_1^1 & \cdots & x_p^1 \\ \vdots & \vdots & \cdots & \vdots \\ 1 & x_1^n & \cdots & x_p^n \end{pmatrix}. \tag{5.5}$$

La somme des moindres carrés s'écrit alors

$$\text{RSS} = \left(\vec{y} - X\vec{\beta}\right)^\top \left(\vec{y} - X\vec{\beta}\right). \tag{5.6}$$

Il s'agit d'une forme quadratique convexe en $\vec{\beta}$, que l'on peut donc minimiser en annulant son gradient $\nabla_{\vec{\beta}}\text{RSS} = -2X^\top \left(\vec{y} - X\vec{\beta}\right)$. On obtient alors

$$X^\top X\vec{\beta}^* = X^\top \vec{y}. \tag{5.7}$$

Théorème 5.1 *Si le rang de la matrice X est égal à son nombre de colonnes, alors la somme des moindres carrés 5.6 est minimisée pour*

$$\vec{\beta}^* = \left(X^\top X\right)^{-1} X^\top \vec{y}.$$

■

Démonstration. Si X est de rang colonne plein, alors $X^\top X$ est inversible. □

Si $X^\top X$ n'est pas inversible, on pourra néanmoins trouver une solution (non unique) pour $\vec{\beta}$ en utilisant à la place de $\left(X^\top X\right)^{-1}$ un pseudo-inverse (par exemple, celui de Moore-Penrose) de $X^\top X$, c'est-à-dire une matrice M telle que $X^\top XMX^\top X = X^\top X$.

Remarque

On peut aussi (et ce sera préférable quand p est grand et que l'inversion de la matrice $X^\top X \in \mathbb{R}^{p\times p}$ est donc coûteuse) obtenir une estimation de $\vec{\beta}$ par un algorithme à directions de descente (voir la section A.3.3).

Attention

On fera attention à ne pas confondre les *variables*, qui sont les p valeurs $x_1, x_2, \ldots, x_p$ qui décrivent les données, et les *paramètres*, qui sont les $p+1$ valeurs $\beta_0, \beta_1, \ldots, \beta_p$ qui paramètrent le modèle.

La régression linéaire produit un modèle interprétable, au sens où les β_j permettent de comprendre l'importance relative des variables sur la prédiction. En effet, plus $|\beta_j|$ est grande, plus la j-ème variable a un effet important sur la prédiction, et le signe de β_j nous indique la direction de cet effet.

Attention ! Cette interprétation n'est valide que si les variables ne sont pas corrélées, et que x_j peut être modifiée sans perturber les autres variables. De plus, si les variables sont corrélées, X n'est pas de rang colonne plein et $X^\top X$ n'est donc pas inversible. Ainsi la régression linéaire admet plusieurs solutions. Intuitivement, on peut passer de l'une à l'autre de ces solutions car une perturbation d'un des poids β_j peut être compensée en modifiant les poids des variables corrélées à x_j.

5.2.3 Théorème de Gauss-Markov

Définition 5.2 (BLUE) Étant donné un vecteur de paramètres ω en q dimensions, et un vecteur $\vec{y}$ à partir duquel l'estimer, on dit que l'estimateur ω^* est le *meilleur estimateur non biaisé* de ω, ou *BLUE* pour *Best Linear Unbiased Estimator*, de ω, si :

1. ω^* est un estimateur linéaire de ω, autrement dit il existe une matrice A telle que $\omega^* = A\vec{y}$;
2. ω^* est non biaisé, autrement dit, $\mathbb{E}[\omega^*] = \omega$
3. Quel que soit l'estimateur linéaire non biaisé $\tilde{\omega}$ de ω, la variance de cet estimateur est supérieure ou égale à celle de ω^*.

■

Théorème 5.2 *Soient n observations $\vec{x}^1, \vec{x}^2, \ldots, \vec{x}^n \in \mathbb{R}^p$ et leurs étiquettes $y^1, y^2, \ldots, y^n \in \mathbb{R}$. Sous l'hypothèse que, pour tout i, $y^i = \beta_0 + \sum_{j=1}^{p} \beta_j x_j^i + \epsilon^i$ et que les ϵ^i sont normalement distribuées et centrées en 0, alors l'estimateur de $\vec{\beta}$ par la méthode des moindres carrés en est l'unique BLUE.* ■

Démonstration. Tout d'abord, $\vec{\beta}^* = \left(X^\top X\right)^{-1} X^\top \vec{y}$ et $\vec{\beta}^*$ est donc *linéaire.*

Son espérance est $\mathbb{E}[\vec{\beta}^*] = \mathbb{E}\left[\left(X^\top X\right)^{-1} X^\top \left(X\vec{\beta} + \epsilon\right)\right]$.

Comme $X, \vec{y}$ et $\vec{\beta}$ ne sont pas aléatoires, et que ϵ a une espérance de 0, on obtient $\mathbb{E}[\vec{\beta}^*] = \mathbb{E}\left[\left(X^\top X\right)^{-1} X^\top X\vec{\beta}\right] = \vec{\beta}$. Ainsi, $\vec{\beta}^*$ est *non biaisé.*

Sa variance vaut :

$$\begin{aligned}\text{Var}(\vec{\beta}^*) &= \text{Var}\left((X^\top X)^{-1}X^\top \vec{y}\right) = \text{Var}\left((X^\top X)^{-1}X^\top \left(X\vec{\beta} + \epsilon\right)\right) \\ &= \text{Var}\left((X^\top X)^{-1}X^\top \epsilon\right) = \sigma^2 (X^\top X)^{-1}.\end{aligned}$$

Enfin, supposons que $\tilde{\beta} = A\vec{y}$ soit un autre estimateur linéaire et non biaisé de $\vec{\beta}$. Par définition, $\mathbb{E}[\tilde{\beta}] = \vec{\beta}$. En remplaçant $\vec{y}$ par sa valeur, on a $\mathbb{E}\left[A\left(X\vec{\beta} + \epsilon\right)\right] = \vec{\beta}$ et ainsi $AX\vec{\beta} = \vec{\beta}$. Comme cela est vrai pour tout $\vec{\beta}$, on en conclut que $AX = I$.

Posons $D = A - (X^\top X)^{-1}X^\top$, de sorte à ce que $\tilde{\beta} - \vec{\beta}^* = D\vec{y}$. La variance de $\tilde{\beta}$ vaut $\text{Var}(\tilde{\beta}) = \text{Var}(A\vec{y}) = \text{Var}(A\epsilon) = AA^\top\sigma^2$.

Nous pouvons exprimer $AA^\top$ de la façon suivante :

$$\begin{aligned}AA^\top &= \left(D + (X^\top X)^{-1}X^\top\right)\left(D + (X^\top X)^{-1}X^\top\right)^\top \\ &= \left(D + (X^\top X)^{-1}X^\top\right)\left(D^\top + X(X^\top X)^{-1}\right) \\ &= DD^\top + DX(X^\top X)^{-1} + (X^\top X)^{-1}X^\top D^\top + (X^\top X)^{-1}.\end{aligned}$$

Comme $AX = I$, $DX = AX - (X^\top X)^{-1}X^\top X = 0$ et donc $X^\top D^\top = 0$ de même.

On obtient donc $AA^\top = DD^\top + (X^\top X)^{-1}$.

Ainsi, $\text{Var}(\tilde{\beta}) = \sigma^2 DD^\top + \sigma^2(X^\top X)^{-1} = \text{Var}(\vec{\beta}^*) + \sigma^2 DD^\top$.

Comme $\sigma^2 > 0$ et $DD^\top$ est semi-définie positive, $\text{Var}(\tilde{\beta}) > \text{Var}(\vec{\beta}^*)$ sauf si $D = 0$, auquel cas $\tilde{\beta} = \vec{\beta}^*$. □

Remarque

L'hypothèse de normalité sur ϵ n'est pas nécessaire : il suffit que les erreurs $\{\epsilon_i\}_{i=1,\ldots,n}$ aient une espérance nulle, aient la même variance finie σ^2 (on parle d'*homoscédasticité*), et ne soient pas corrélées entre elles ($\text{Cov}(\epsilon_i, \epsilon_l) = 0$ pour $i \neq l$).

5.3 Régression logistique

Supposons maintenant que nous voulions résoudre un problème de classification binaire de manière linéaire, c'est-à-dire modéliser $y \in \{0, 1\}$ à l'aide d'une combinaison linéaire de variables.

Il ne semble pas raisonnable de modéliser directement y comme une combinaison linéaire de plusieurs variables réelles : une telle combinaison est susceptible de prendre non pas deux mais une infinité de valeurs.

Nous pourrions alors envisager un modèle probabiliste, dans lequel $\mathbb{P}(Y = y|X = \vec{x})$ soit modélisé par une combinaison linéaire des variables de $\vec{x}$. Cependant, $\mathbb{P}(Y = y|X = \vec{x})$ doit être comprise entre 0 et 1, et intuitivement, cette fonction n'est pas linéaire : si $\mathbb{P}(Y = 0|X = \vec{x})$ est très proche de 1, autrement dit qu'il est très probable que $\vec{x}$ est négative, une petite perturbation de $\vec{x}$ ne doit pas beaucoup affecter cette probabilité ; mais à l'inverse, si $\mathbb{P}(Y = 0|X = \vec{x})$ est très proche de 0.5, autrement dit que l'on est très peu certain de l'étiquette de $\vec{x}$, rien ne s'oppose à ce qu'une petite perturbation de $\vec{x}$ n'affecte cette probabilité. C'est pour cela qu'il est classique de modéliser une *transformation logit* de $\mathbb{P}(Y = y|X = \vec{x})$ comme une combinaison linéaire des variables.

Définition 5.3 (Fonction logit) On appelle *fonction logit* la fonction

$$\begin{aligned} \text{logit} : [0,1] &\to \mathbb{R} \\ p &\mapsto \log \frac{p}{1-p} \end{aligned}$$

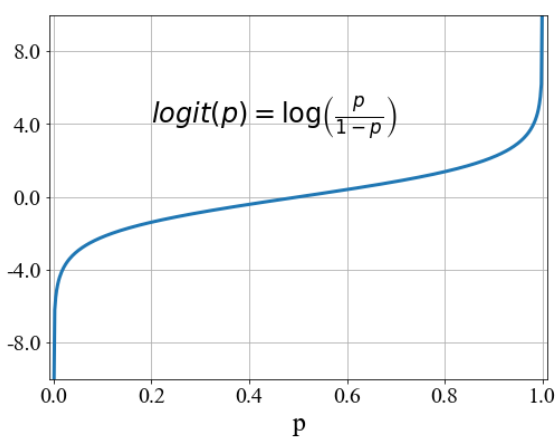
■

La fonction logit (figure 5.2a) est la réciproque de la fonction logistique (figure 5.2b).

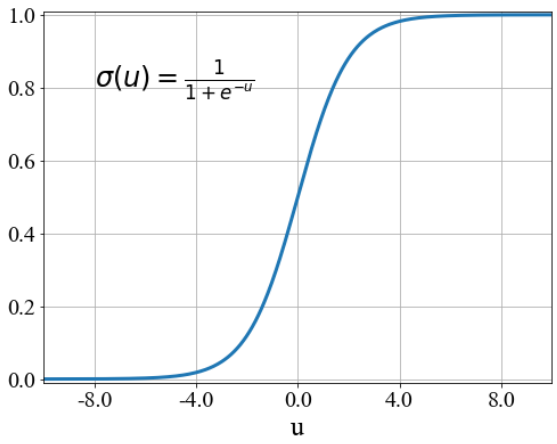
Définition 5.4 (Fonction logistique) On appelle *fonction logistique* la fonction

$$\begin{aligned} \sigma : \mathbb{R} &\to [0,1] \\ u &\mapsto \frac{1}{1+e^{-u}} = \frac{e^u}{1+e^u}. \end{aligned}$$

Attention à ne pas confondre la fonction logistique avec l'écart-type, tous deux étant couramment notés σ. ■



(A) Fonction logit.



(B) Fonction logistique.

FIGURE 5.2 – Fonctions logit et logistique.

5.3.1 Formulation

Ainsi, nous cherchons donc à modéliser $\log \frac{\mathbb{P}(Y=1|\vec{x})}{1-\mathbb{P}(Y=1|\vec{x})}$ comme la combinaison linéaire $\vec{\beta}^\top \vec{x}$, où, de manière équivalente, $\mathbb{P}(Y = 1|\vec{x})$ comme $\sigma(\vec{\beta}^\top \vec{x})$. Nous utilisons ici la transformation 5.5 de $\vec{x}$.

Étant données n observations $\mathcal{D} = \{\vec{x}^{\,i}, y^i\}_{i=1,\dots,n}$, que l'on suppose iid, le log de la vraisemblance de $\vec{\beta}$ est

$$\begin{aligned}\log \mathbb{P}(\mathcal{D}|\vec{\beta}) &= \log \prod_{i=1}^{n} \mathbb{P}(X = \vec{x}^{\,i}, Y = y^i|\vec{\beta}) = \log \prod_{i=1}^{n} \mathbb{P}(Y = y^i|\vec{x}^{\,i}, \vec{\beta}) + \log \prod_{i=1}^{n} \mathbb{P}(X = \vec{x}^{\,i}) \\ &= \sum_{i=1}^{n} \log \mathbb{P}(Y = 1|\vec{x}^{\,i}, \vec{\beta})^{y^i} \left(1 - \mathbb{P}(Y = 1|\vec{x}^{\,i}, \vec{\beta})^{1-y^i}\right) + \mathcal{C} \qquad (5.8) \\ &= \sum_{i=1}^{n} y^i \log \sigma(\vec{\beta}^{\top}\vec{x}^{\,i}) + (1 - y^i) \log \left(1 - \sigma(\vec{\beta}^{\top}\vec{x}^{\,i})\right) + \mathcal{C},\end{aligned}$$

où $\mathcal{C}$ est une constante par rapport à $\vec{\beta}$. Nous cherchons à maximiser cette vraisemblance.

Définition 5.5 (Régression logistique) On appelle *régression logistique* le modèle $f : x \mapsto \sigma(\vec{\beta}^{\top}\vec{x})$ dont les coefficients sont obtenus par

$$\arg\max_{\vec{\beta} \in \mathbb{R}^{p+1}} \sum_{i=1}^{n} y^i \log \sigma(\vec{\beta}^{\top}\vec{x}^{\,i}) + (1 - y^i) \log \left(1 - \sigma(\vec{\beta}^{\top}\vec{x}^{\,i})\right). \qquad (5.9)$$

■

Maximiser la vraisemblance de β sous ce modèle est équivalent à minimiser le risque empirique défini en utilisant la fonction de coût logistique (voir section 2.4.1).

5.3.2 Solution

La vraisemblance de la régression logistique 5.8 est une fonction concave.

Théorème 5.3 *Le gradient en $\vec{\beta}$ du maximum de vraisemblance de la régression logistique vaut*

$$\sum_{i=1}^{n} \left(y^i - \frac{1}{1 + e^{-\beta^{\top}\vec{x}^{\,i}}}\right) \vec{x}^{\,i}.$$

■

Démonstration. Ce résultat s'obtient en utilisant une propriété remarquable de la fonction logistique : $\sigma'(u) = \sigma(u)(1 - \sigma(u))$. □

Ce gradient ne peut pas être annulé de manière analytique et la régression logistique n'admet donc pas de solution explicite. On trouvera généralement sa solution par l'algorithme du gradient ou une de ses variantes. Ces algorithmes convergent vers la solution optimale car la vraisemblance est concave et n'admet donc pas de maximum local.

5.4 Régression polynomiale

Dans le cas de la régression polynomiale de degré d, on cherche maintenant une fonction de décision de la forme suivante :

$$f : \vec{x} \mapsto \beta_{00} + \sum_{j=1}^{p} \beta_{1j} x_j + \sum_{j=1}^{p} \beta_{2j} x_j^2 + \cdots + \sum_{j=1}^{p} \beta_{dj} x_j^d. \qquad (5.10)$$

Il s'agit en fait d'une régression linéaire sur $p \times d$ variables : $x_1, x_2, \ldots, x_p, x_1^2, x_2^2, \ldots, \vec{x}_p^2, \ldots, x_1^d, x_2^d, \ldots x_p^d$.

Points clefs

- On peut apprendre les coefficients d'un modèle de régression paramétrique par maximisation de vraisemblance, ce qui équivaut à minimiser le risque empirique en utilisant le coût quadratique comme fonction de perte, et revient à la méthode des moindres carrés.
- La régression linéaire admet une unique solution $\vec{\beta}^* = \left(X^\top X\right)^{-1} X^\top \vec{y}$ si et seulement si $X^\top X$ est inversible. Dans le cas contraire, il existe une infinité de solutions.
- La régression polynomiale se ramène à une régression linéaire.
- Pour un problème de classification binaire, utiliser le coût logistique comme fonction de perte dans la minimisation du risque empirique revient à maximiser la vraisemblance d'un modèle dans lequel la probabilité d'appartenir à la classe positive est la transformée logit d'une combinaison linéaire des variables.
- La régression logistique n'a pas de solution analytique, mais on peut utiliser un algorithme à directions de descente pour apprendre le modèle.

Pour aller plus loin

- La régression logistique peut naturellement être étendue à la classification multi-classe en utilisant l'entropie croisée comme fonction de coût.
- Nous avons vu comment construire des modèles polynomiaux en appliquant une transformation polynomiale aux variables d'entrées. Ce n'est pas la seule transformation que l'on puisse leur appliquer, et l'on peut aussi considérer par exemple des fonctions polynomiales par morceaux (ou splines) ou des ondelettes. Cet ensemble de techniques est connu sous le nom de *basis expansions* en anglais. On se référera par exemple au chapitre 5 de Hastie et al. (2009). Cet ouvrage dans son ensemble permet par ailleurs d'approfondir les notions abordées dans ce chapitre.

Bibliographie

Hastie, T., Tibshirani, R., et Friedman, J. (2009). *The Elements of Statistical Learning : Data Mining, Inference, and Prediction, Second Edition*. Springer-Verlag, New York, 2nd edition. `https://web.stanford.edu/~hastie/ElemStatLearn/`

Chapitre 6

Régularisation

Lorsque les variables explicatives sont corrélées, ou trop nombreuses, la complexité d'un modèle de régression linéaire est bien souvent trop élevée, ce qui conduit à une situation de sur-apprentissage.

Dans ce chapitre, nous verrons comment *régulariser* ces modèles en contrôlant les coefficients de régression, c'est-à-dire les poids affectés à chacune des variables dans leur combinaison linéaire. Nous présenterons le cas de la régression linéaire, mais les techniques présentées dans ce chapitre s'appliquent plus largement au cas de la régression logistique.

Objectifs

- Contrôler la complexité d'un modèle par la régularisation
- Définir le lasso, la régression ridge, et elastic net
- Comprendre le rôle des normes ℓ_1 et ℓ_2 comme régulariseurs
- Choisir un coefficient de régularisation.

6.1 Qu'est-ce que la régularisation ?

Lorsque les variables sont fortement corrélées, ou que leur nombre dépasse celui des observations, la matrice $X \in \mathbb{R}^{p+1}$ représentant nos données ne peut pas être de rang colonne plein. Ainsi, la matrice $X^\top X$ n'est pas inversible et il n'existe pas de solution unique à une régression linéaire par minimisation des moindres carrés. Il y a donc un risque de sur-apprentissage : le modèle n'étant pas unique, comment peut-on garantir que c'est celui que l'on a sélectionné qui généralise le mieux ?

Pour limiter ce risque de sur-apprentissage, nous allons chercher à contrôler simultanément l'erreur du modèle sur le jeu d'entraînement et les valeurs des coefficients de régression affectés à chacune des variables. Contrôler ces coefficients est une façon de contrôler la complexité du modèle : comme nous le verrons par la suite, ce contrôle consiste à contraindre les coefficients à appartenir à un sous-ensemble strict de $\mathbb{R}^{p+1}$ plutôt que de pouvoir prendre n'importe quelle valeur dans cet espace, ce qui restreint l'espace des solutions possibles.

Définition 6.1 (Régularisation) On appelle *régularisation* le fait d'apprendre un modèle en minimisant la somme du risque empirique sur le jeu d'entraînement et d'un terme de contrainte Ω sur les solutions possibles :

$$f = \underset{h \in \mathcal{F}}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} L(h(\vec{x}^{\,i}), y^i) + \lambda \Omega(h), \tag{6.1}$$

où le coefficient de régularisation $\lambda \in \mathbb{R}_+$ contrôle l'importance relative de chacun des termes. ■

Dans le cas d'un modèle de régression linéaire, nous allons utiliser comme fonction de perte la somme des moindres carrés. Les régulariseurs que nous allons voir sont fonction du vecteur de coefficients de régression $\vec{\beta}$:

$$\underset{\vec{\beta} \in \mathbb{R}^{p+1}}{\arg\min} \left(\vec{y} - X\vec{\beta}\right)^\top \left(\vec{y} - X\vec{\beta}\right) + \lambda \Omega(\vec{\beta})$$

ou, de manière équivalente,

$$\underset{\vec{\beta} \in \mathbb{R}^{p+1}}{\arg\min} \left|\left|\vec{y} - X\vec{\beta}\right|\right|_2^2 + \lambda \Omega(\vec{\beta}). \tag{6.2}$$

Nous utilisons ici la transformation 5.5 de $\vec{x}$ qui consiste à ajouter à la matrice de design X une colonne de 1 pour simplifier les notations.

Encart

Coefficient de régularisation

Le coefficient de régularisation λ est un hyperparamètre de la régression linéaire régularisée.

Quand λ tend vers $+\infty$, le terme de régularisation prend de plus en plus d'importance, jusqu'à ce qu'il domine le terme d'erreur et que seule compte la minimisation du régulariseur. Dans la plupart des cas, le régulariseur est minimisé quand $\vec{\beta} = \vec{0}$, et il n'y a plus d'apprentissage.

À l'inverse, quand λ tend vers 0, le terme de régularisation devient négligeable devant le terme d'erreur, et $\vec{\beta}$ prendra comme valeur une solution de la régression linéaire non régularisée.

Comme tout hyperparamètre, λ peut être choisi par validation croisée. On utilisera généralement une grille de valeurs logarithmique.

6.2 La régression ridge

Une des formes les plus courantes de régularisation, utilisée dans de nombreux domaines faisant intervenir des problèmes inverses mal posés, consiste à utiliser comme régulariseur la norme ℓ_2 du vecteur $\vec{\beta}$:

$$\Omega_{\text{ridge}}(\vec{\beta}) = \left|\left|\vec{\beta}\right|\right|_2^2 = \sum_{j=0}^{p} \beta_j^2. \tag{6.3}$$

6.2.1 Formulation de la régression ridge

Définition 6.2 (Régression ridge) On appelle *régression ridge* le modèle $f : x \mapsto \vec{\beta}^{\,\top} \vec{x}$ dont les coefficients sont obtenus par

$$\underset{\vec{\beta} \in \mathbb{R}^{p+1}}{\arg\min} \left|\left|\vec{y} - X\vec{\beta}\right|\right|_2^2 + \lambda \left|\left|\vec{\beta}\right|\right|_2^2. \tag{6.4}$$

■

La régression ridge est un cas particulier de *régularisation de Tikhonov* (développée pour résoudre des équations intégrales). Elle intervient aussi dans les réseaux de neurones, où elle est appelée *weight decay* (dégradation / modération des pondérations, voir section 7.2.4).

6.2.2 Solution

Le problème 6.4 est un problème d'optimisation convexe (voir section A) : il s'agit de minimiser une forme quadratique. Il se résout en annulant le gradient en $\vec{\beta}$ de la fonction objective :

$$\nabla_{\vec{\beta}} \left(\left|\left| \vec{y} - X\vec{\beta} \right|\right|_2^2 + \lambda \left|\left| \vec{\beta} \right|\right|_2^2 \right) = 0 \tag{6.5}$$

En notant $I_p \in \mathbb{R}^{p \times p}$ la matrice identité en dimension p, on obtient :

$$\left(\lambda I_p + X^\top X \right) \vec{\beta}^* = X^\top \vec{y}. \tag{6.6}$$

Comme $\lambda > 0$, la matrice $\lambda I_p + X^\top X$ est toujours inversible. Notre problème admet donc toujours une unique solution explicite. La régularisation par la norme ℓ_2 a permis de transformer un problème potentiellement mal posé en un problème bien posé, dont la solution est :

$$\vec{\beta}^* = \left(\lambda I_p + X^\top X \right)^{-1} X^\top \vec{y}. \tag{6.7}$$

Remarque

Si l'on multiplie la variable x_j par une constante α, le coefficient correspondant dans la régression linéaire non régularisée est divisé par α. En effet, si on appelle X^* la matrice obtenue en remplaçant x_j par αx_j dans X, la solution $\vec{\beta}^*$ de la régression linéaire correspondante vérifie $X^* \left(\vec{y} - X^* \vec{\beta}^* \right) = 0$, tandis que la solution $\vec{\beta}$ de la régression linéaire sur X vérifie $X \left(\vec{y} - X\vec{\beta} \right) = 0$. Ainsi, changer l'échelle d'une variable a comme seul impact sur la régression linéaire non régularisée d'ajuster le coefficient correspondant de manière inversement proportionnelle.

À l'inverse, dans le cas de la régression ridge, remplacer x_j par αx_j affecte aussi le terme de régularisation, et a un effet plus complexe. L'échelle relative des différentes variables peut donc fortement affecter la régression ridge. Il est ainsi recommandé de *standardiser* les variables avant l'apprentissage, c'est-à-dire de toutes les ramener à avoir un écart-type de 1 en les divisant par leur écart-type :

$$x_j^i \leftarrow \frac{x_j^i}{\sqrt{\frac{1}{n} \sum_{i=1}^n \left(x_j^i - \frac{1}{n} \sum_{i=1}^n x_j^i \right)^2}} \tag{6.8}$$

Attention : pour éviter le sur-apprentissage, il est important que cet écart-type soit calculé sur le jeu d'entraînement uniquement, puis appliqué ensuite aux jeux de test ou validation.

La régression ridge a un effet de « regroupement » sur les variables corrélées, au sens où des variables corrélées auront des coefficients similaires.

6.2.3 Chemin de régularisation

Définition 6.3 (Chemin de régularisation) On appelle *chemin de régularisation* l'évolution de la valeur du coefficient de régression d'une variable en fonction du coefficient de régularisation λ. ■

Le chemin de régularisation permet de comprendre l'effet de la régularisation sur les valeurs de $\vec{\beta}$. En voici un exemple sur la figure 6.1.

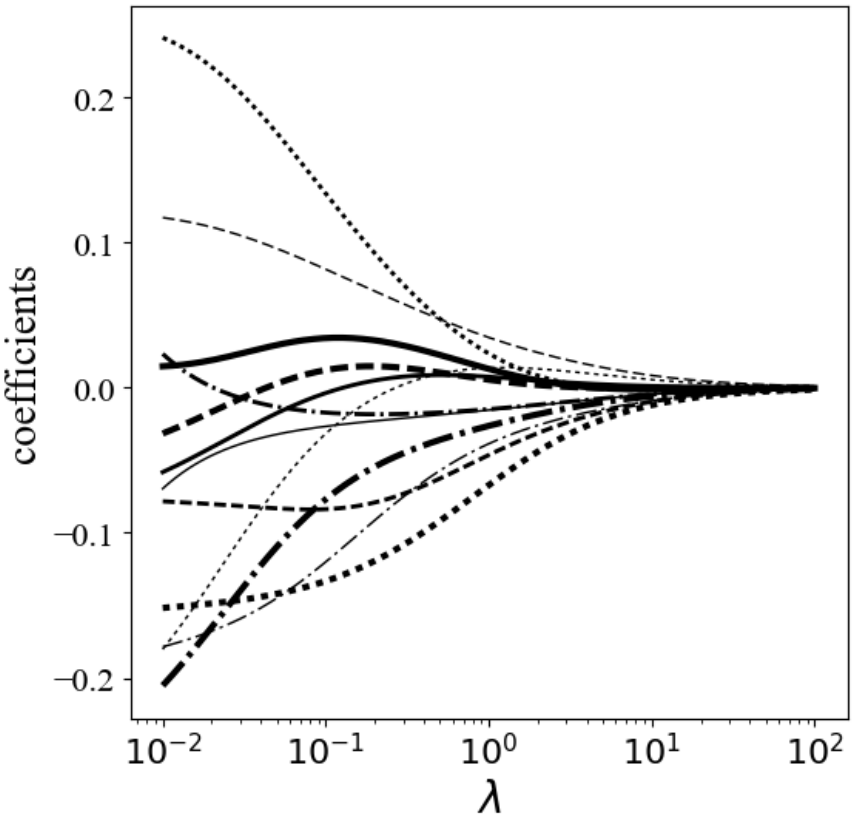


FIGURE 6.1 – Chemin de régularisation de la régression ridge pour un jeu de données avec 12 variables. Chaque ligne représente l'évolution du coefficient de régression d'une de ces variables quand λ augmente : le coefficient évolue de sa valeur dans la régression non régularisée vers 0.

6.2.4 Interprétation géométrique

Théorème 6.1 *Étant donnés $\lambda \in \mathbb{R}_+$, $X \in \mathbb{R}^{n\times p}$ et $\vec{y} \in \mathbb{R}^n$, il existe un unique $t \in \mathbb{R}_+$ tel que le problème 6.4 soit équivalent à*

$$\underset{\vec{\beta}\in\mathbb{R}^{p+1}}{\arg\min} \left|\left|\vec{y} - X\vec{\beta}\right|\right|_2^2 \text{ tel que } \left|\left|\vec{\beta}\right|\right|_2^2 \leq t. \tag{6.9}$$

■

Démonstration. L'équivalence s'obtient par dualité et en écrivant les conditions de Karun-Kush-Tucker. □

La régression ridge peut donc être formulée comme un problème d'optimisation quadratique (minimiser $\left|\left|\vec{y} - X\vec{\beta}\right|\right|_2^2$) sous contraintes ($\left|\left|\vec{\beta}\right|\right|_2^2 \leq t$) : la solution doit être contenue dans la boule ℓ_2 de rayon $\sqrt{t}$. Sauf dans le cas où l'optimisation sans contrainte vérifie déjà la condition, cette solution sera sur la frontière de cette boule, comme illustré sur la figure 6.2.

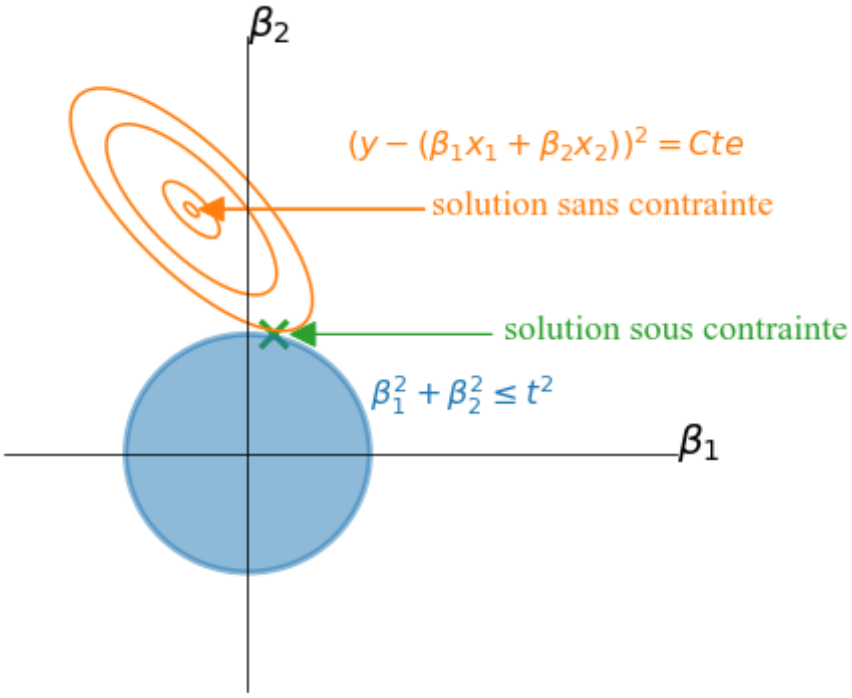


FIGURE 6.2 – La solution du problème d'optimisation sous contraintes 6.9 (ici en deux dimensions) se situe sur une ligne de niveau de la somme des moindres carrés tangente à la boule ℓ_2 de rayon $\sqrt{t}$.

6.3 Le lasso

6.3.1 Parcimonie

Dans certaines applications, il peut être raisonnable de supposer que l'étiquette que l'on cherche à prédire n'est expliquée que par un nombre restreint de variables. Il est dans ce cas souhaitable d'avoir un modèle *parcimonieux*, ou *sparse*, c'est-à-dire dans lequel un certain nombre de coefficients sont nuls : les variables correspondantes peuvent être retirées du modèle.

Pour ce faire, Robert Tibshirani a proposé en 1996 d'utiliser comme régulariseur la norme ℓ_1 du coefficient $\vec{\beta}$:

$$\Omega_{\text{lasso}}(\vec{\beta}) = \left|\left|\vec{\beta}\right|\right|_1 = \sum_{j=0}^{p} |\beta_j|. \tag{6.10}$$

Pour comprendre pourquoi ce régulariseur permet de « pousser » certains coefficients vers 0, on se reportera à la section 6.3.4 et à la figure 6.3.

6.3.2 Formulation du lasso

Définition 6.4 (Lasso) On appelle *lasso* le modèle $f : x \mapsto \vec{\beta}^{\top}\vec{x}$ dont les coefficients sont obtenus par

$$\underset{\vec{\beta}\in\mathbb{R}^{p+1}}{\arg\min} \left|\left|\vec{y} - X\vec{\beta}\right|\right|_2^2 + \lambda \left|\left|\vec{\beta}\right|\right|_1. \tag{6.11}$$

■

Le nom de lasso est en fait un acronyme, pour *Least Absolute Shrinkage and Selection Operator* : il s'agit d'une méthode qui utilise les valeurs *absolues* des coefficients (la norme ℓ_1) pour réduire (*shrink*) ces coefficients, ce qui permet de *sélectionner* les variables qui n'auront pas un coefficient nul. En traitement du signal, le lasso est aussi connu sous le nom de *poursuite de base* (*basis pursuit* en anglais).

Remarque

En créant un modèle parcimonieux et en permettant d'éliminer les variables ayant un coefficient nul, le lasso est une méthode de sélection de variable supervisée. Il s'agit donc aussi d'une méthode de réduction de dimension.

6.3.3 Solution

Le lasso 6.11 n'admet pas de solution explicite. On pourra utiliser un algorithme à directions de descente (voir section A.3.3) pour le résoudre. De plus, il ne s'agit pas toujours d'un problème strictement convexe (en particulier, quand $p > n$) et il n'admet donc pas nécessairement une unique solution. En pratique, cela pose surtout problème quand les variables ne peuvent pas être considérées comme les réalisations de lois de probabilité continues. Néanmoins, il est possible de montrer que les coefficients non nuls dans deux solutions ont nécessairement le même signe. Ainsi, l'effet d'une variable a la même direction dans toutes les solutions qui la considèrent, ce qui facilite l'interprétation d'un modèle appris par le lasso.

6.3.4 Interprétation géométrique

Comme précédemment, le problème 6.11 peut être reformulé comme un problème d'optimisation quadratique sous contraintes :

Théorème 6.2 *Étant donnés $\lambda \in \mathbb{R}_+$, $X \in \mathbb{R}^{n\times p}$ et $\vec{y} \in \mathbb{R}^n$, il existe un unique $t \in \mathbb{R}_+$ tel que le problème 6.11 soit équivalent à*

$$\underset{\vec{\beta}\in\mathbb{R}^{p+1}}{\arg\min} \left|\left|\vec{y} - X\vec{\beta}\right|\right|_2^2 \text{ tel que } \left|\left|\vec{\beta}\right|\right|_1 \leq t. \tag{6.12}$$

■

La solution doit maintenant être contenue dans la boule ℓ_1 de rayon t. Comme cette boule a des « coins », les lignes de niveau de la forme quadratique sont plus susceptibles d'y être tangente en un point où une ou plusieurs coordonnées sont nulles (voir figure 6.3).

6.3.5 Chemin de régularisation

Sur le chemin de régularisation du lasso (par exemple figure 6.4, sur les mêmes données que pour la figure 6.1), on observe que les variables sortent du modèle les unes après les autres, jusqu'à ce que tous les coefficients soient nuls. On remarquera aussi que le chemin de régularisation pour n'importe quelle variable est linéaire par morceaux ; c'est une propriété du lasso.

Remarque

(Stabilité)

Si plusieurs variables corrélées contribuent à la prédiction de l'étiquette, le lasso va avoir tendance à choisir une seule d'entre elles (affectant un poids de 0 aux autres), plutôt que de répartir les poids équitablement comme la régression ridge. C'est ainsi qu'on arrive à avoir des modèles très parcimonieux. Cependant,

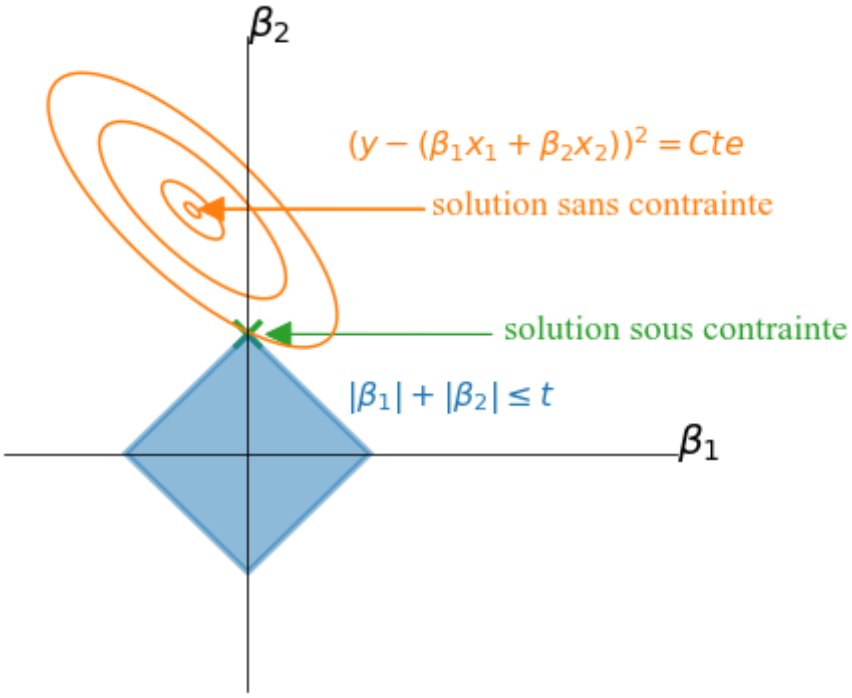


FIGURE 6.3 – La solution du problème d'optimisation sous contraintes 6.12 (ici en deux dimensions) se situe sur une ligne de niveau de la somme des moindres carrés tangente à la boule ℓ_1 de rayon t.

le choix de cette variable est aléatoire, et peut changer si l'on répète la procédure d'optimisation. Le lasso a donc tendance à être instable.

6.4 Elastic net

La régularisation par la norme ℓ_1 permet d'obtenir un modèle parcimonieux et donc plus facilement interprétable, tandis que la régularisation par la norme ℓ_2 permet elle d'éviter le sur-apprentissage et de grouper les variables corrélées. Il est assez naturel de souhaiter combiner les deux normes dans le régulariseur suivant :

$$\Omega_{\text{enet}}(\vec{\beta}) = \left((1-\alpha) \left|\left|\vec{\beta}\right|\right|_1 + \alpha \left|\left|\vec{\beta}\right|\right|_2^2 \right). \tag{6.13}$$

Ce régulariseur est paramétré par $\alpha \in [0,1]$. Quand $\alpha = 0$, on retrouve la régularisation ℓ_1, et quand $\alpha = 1$, la régularisation ℓ_2.

Définition 6.5 (Elastic net) On appelle *elastic net* le modèle $f : x \mapsto \vec{\beta}^\top \vec{x}$ dont les coefficients sont obtenus par

$$\underset{\vec{\beta}\in\mathbb{R}^{p+1}}{\arg\min} \left|\left|\vec{y} - X\vec{\beta}\right|\right|_2^2 + \lambda \left((1-\alpha) \left|\left|\vec{\beta}\right|\right|_1 + \alpha \left|\left|\vec{\beta}\right|\right|_2^2 \right). \tag{6.14}$$

■

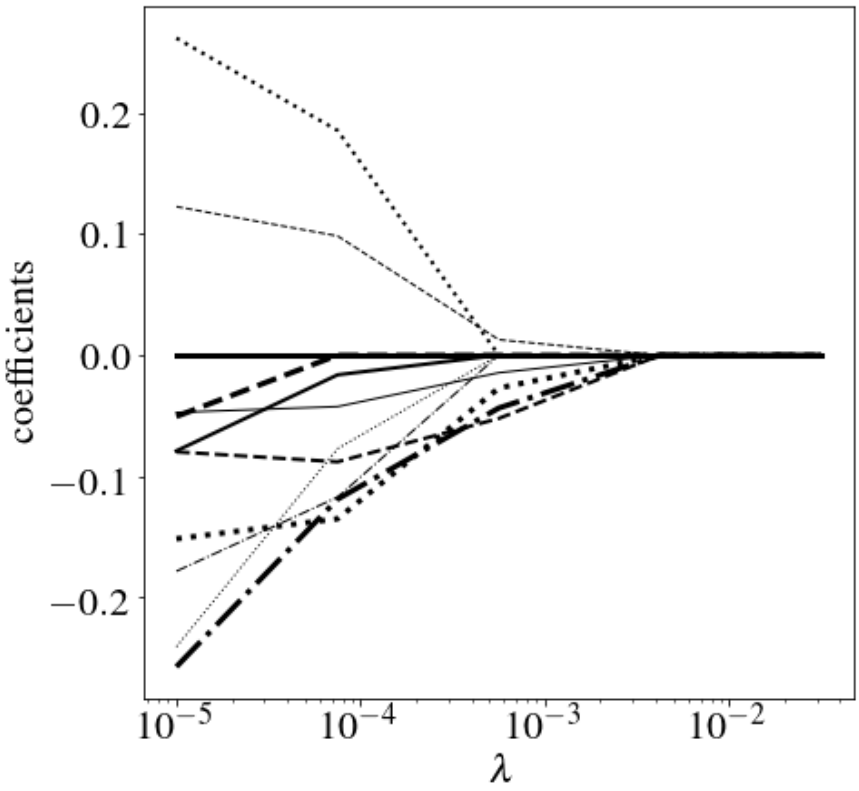


FIGURE 6.4 – Chemin de régularisation du lasso pour un jeu de données avec 12 variables. Chaque ligne représente l'évolution du coefficient de régression d'une de ces variables quand λ augmente : les variables sont éliminées les unes après les autres.

On obtient la solution de l'elastic net par un algorithme à directions de descente. Cette solution est parcimonieuse, mais moins que celle du lasso. En effet, quand plusieurs variables fortement corrélées sont pertinentes, le lasso sélectionnera une seule d'entre elles, tandis que, par effet de la norme ℓ_2, l'elastic net les sélectionnera toutes et leur affectera le même coefficient.

Points clefs

— Ajouter un terme de régularisation, fonction du vecteur des coefficients $\vec{\beta}$, au risque empirique de la régression linéaire permet d'éviter le sur-apprentissage

— La régression ridge utilise la norme ℓ_2 de $\vec{\beta}$ comme régulariseur ; elle admet toujours une unique solution analytique, et a un effet de regroupement sur les variables corrélées.

— Le lasso utilise la norme ℓ_1 de $\vec{\beta}$ comme régulariseur ; il crée un modèle parcimonieux, et permet donc d'effectuer une réduction de dimension supervisée.

— De nombreux autres régulariseurs sont possibles en fonction de la structure du problème.

Pour aller plus loin

- Au-delà des normes ℓ_1 et ℓ_2, il est possible d'utiliser des régulariseurs de la forme $\Omega_{\ell_q}(\vec{\beta}) = ||\vec{\beta}||_q^q$.

- Une famille de régulariseurs appelés « structurés » permettent de sélectionner des variables qui respectent une structure (graphe, groupes, ou arbre) donnée a priori. Ces approches sont utilisées en particulier dans des applications bio-informatiques, par exemple quand on cherche à construire des modèles parcimonieux basés sur l'expression de gènes sous l'hypothèse que seul un petit nombre de voies métaboliques (groupes de gènes) est pertinent. Pour plus de détails, on se référera à l'article de Huang et al. (2011).
- En ce qui concerne l'unicité de la solution du lasso, on se reportera à l'article de Tibshirani (2013).
- L'ouvrage de Hastie et al. (2015) est entièrement consacré au lasso et ses généralisations.

Bibliographie

Hastie, T., Tibshirani, R., et Wainwright, M. (2015). *Statistical Learning with Sparsity : The Lasso and Generalizations*. CRC Press. `http://web.stanford.edu/~hastie/StatLearnSparsity/`.

Huang, J., Zhang, T., et Metaxas, D. (2011). Learning with structured sparsity. *Journal of Machine Learning Research*, 12 :3371–3412.

Tibshirani, R. J. (2013). The lasso problem and uniqueness. *Electronic Journal of Statistics*, 7 :1456–1490.

Chapitre 7

Réseaux de neurones artificiels

De l'annotation automatique d'images à la reconnaissance vocale, en passant par des ordinateurs capables de battre des champions de go et par les véhicules autonomes, les récents succès de l'intelligence artificielle sont nombreux à reposer sur les réseaux de neurones profonds, et le *deep learning* (ou apprentissage profond) fait beaucoup parler de lui.

Les réseaux de neurones artificiels ne sont au fond rien d'autre que des modèles paramétriques, potentiellement complexes : contrairement à la régression linéaire, ils permettent de construire facilement des modèles très flexibles.

Dans ce chapitre, nous aborderons les principes de base d'une classe particulière de réseaux de neurones artificiels, les perceptrons multi-couche, et de leur entraînement. Nous effleurerons la question des réseaux de neurones profonds, que nous considérons comme un sujet avancé n'ayant pas sa place dans cet ouvrage.

Objectifs

— Écrire la fonction de décision correspondant à un perceptron uni- ou multi-couche

— Implémenter la procédure d'apprentissage d'un perceptron

— Expliciter les actualisations d'un perceptron multi-couche dans une procédure de rétropropagation

— Connaître quelques grands principes de la conception et de l'entraînement d'un perceptron multi-couche

— Appréhender les enjeux du *deep learning*.

7.1 Le perceptron

L'histoire des réseaux de neurones artificiels remonte aux années 1950 et aux efforts de psychologues comme Franck Rosenblatt pour comprendre le cerveau humain. Initialement, ils ont été conçus dans le but de modéliser mathématiquement le traitement de l'information par les réseaux de neurones biologiques qui se trouvent dans le cortex des mammifères. De nos jours, leur réalisme biologique importe peu et c'est leur efficacité à modéliser des relations complexes et non linéaires qui fait leur succès.

Le premier réseau de neurones artificiels est le *perceptron* (Rosenblatt, 1957). Loin d'être profond, il comporte une seule couche et a une capacité de modélisation limitée.

7.1.1 Modèle

Le perceptron (figure 7.1) est formé d'une couche d'entrée de p neurones, ou *unités*, correspondant chacune à une variable d'entrée. Ces neurones transmettent la valeur de leur entrée à la couche suivante. À ces p neurones on rajoute généralement une unité de biais, qui transmet toujours la valeur 1. Cette unité correspond à la colonne de 1 que nous avons ajoutée aux données dans les modèles linéaires (équation 5.5). On remplacera dans ce qui suit tout vecteur $\vec{x} = (x_1, x_2, \ldots, x_p)$ par sa version augmentée d'un 1 : $\vec{x} = (1, x_1, x_2, \ldots, x_p)$.

La première et unique couche du perceptron (après la couche d'entrée) contient un seul neurone, auquel sont connectées toutes les unités de la couche d'entrée.

Ce neurone calcule une combinaison linéaire $o(\vec{x}) = w_0 + \sum_{j=1}^{p} w_j x_j$ des signaux $x_1, x_2, \ldots, x_p$ qu'il reçoit en entrée, auquel il applique une *fonction d'activation* a, dont il transmet en sortie le résultat. Cette sortie met en œuvre la fonction de décision du perceptron.

Ainsi, si l'on appelle w_j le poids de connexion entre l'unité d'entrée j et le neurone de sortie, ce neurone calcule

$$f(\vec{x}) = a(o(\vec{x})) = a\left(w_0 + \sum_{j=1}^{p} w_j x_j\right) = a\left(\langle \vec{w}, \vec{x} \rangle\right). \tag{7.1}$$

Il s'agit donc bien d'un modèle paramétrique.

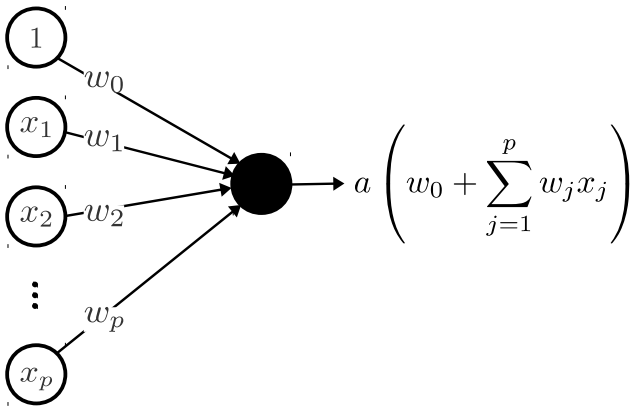


FIGURE 7.1 – Architecture d'un perceptron.

Fonctions d'activation

Dans le cas d'un problème de régression, on utilisera tout simplement l'identité comme fonction d'activation. Dans le cas d'un problème de classification binaire, on pourra utiliser :

— Pour prédire directement une étiquette binaire, une fonction de seuil :

$$f : \vec{x} \mapsto \begin{cases} 0 & \text{si } o(\vec{x}) \leq 0 \\ 1 & \text{sinon.} \end{cases} \tag{7.2}$$

— Pour prédire la probabilité d'appartenir à la classe positive, comme dans le cas de la régression logistique (section 5.3), une fonction logistique :

$$f : \vec{x} \mapsto \frac{1}{1 + e^{o(\vec{x})}} = \frac{1}{1 + \exp\left(\langle \vec{w}, \vec{x} \rangle\right)}. \tag{7.3}$$

Classification multi-classe

Dans le cas d'un problème de classification multi-classe, on modifiera l'architecture du perceptron de sorte à n'avoir non plus 1 mais C neurones dans la couche de sortie, où C est le nombre de classes. Les $p+1$ neurones de la couche d'entrée seront ensuite tous connectés à chacun de ces neurones de sortie (on aura donc $(p+1)C$ poids de connexion, notés w_j^c.) Cette architecture est illustrée sur la figure 7.2.

On utilise alors comme fonction d'activation la fonction softmax :

Définition 7.1 (Fonction softmax) On appelle *fonction softmax*, ou *fonction exponentielle normalisée*, la fonction $\sigma : \mathbb{R}^C \to [0,1]^C$ définie par :

$$\sigma(o_1, o_2, \ldots, o_C)_c = \frac{e^{o_c}}{\sum_{k=1}^{C} e^{o_k}}.$$

■

Dans le cas d'un perceptron classique, o_k est la combinaison linéaire calculée au k-ème neurone de sortie : $o_k = \langle \vec{w}^{\,k}, \vec{x} \rangle$.

La fonction softmax généralise la fonction logistique utilisée dans le cas binaire. Elle produit C nombres réels positifs de somme 1, et peut être considérée comme une version régulière de la fonction $\arg\max$: quand $o_c > o_{k \neq c}$, $\sigma(\vec{o})_c \approx 1$ et $\sigma(\vec{o})_{k \neq c} \approx 0$.

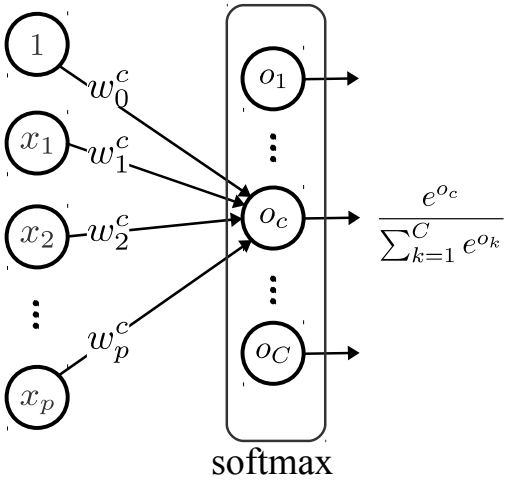


FIGURE 7.2 – Architecture d'un perceptron multi-classe.

7.1.2 Entraînement

Pour entraîner un perceptron, nous allons chercher, comme par exemple pour la régression paramétrique, à minimiser le risque empirique. Cependant, nous allons supposer que les observations $(\vec{x}^{\,i}, y^i)$ ne sont pas disponibles simultanément, mais qu'elles sont observées séquentiellement. Cette hypothèse découle de la plasticité des réseaux de neurones biologiques : ils s'adaptent constamment en fonction des

signaux qu'ils reçoivent. Nous allons donc utiliser un algorithme d'entraînement *incrémental*, qui s'adapte à des observations arrivant les unes après les autres.

Définition 7.2 (Apprentissage incrémental vs hors-ligne) Un algorithme d'apprentissage qui opère sur un unique ensemble de n observations est appelé *hors-ligne*. En anglais, on parlera de *batch learning*.

Par contraste, un algorithme d'apprentissage qui effectue une ou plusieurs opérations à chaque nouvelle observation qui lui est fournie est appelé *incrémental* ou *en ligne*. En anglais, on parlera de *online learning*. ■

Pour minimiser le risque empirique de manière itérative, nous allons nous appuyer sur l'algorithme du gradient (voir section A.3.3). L'algorithme commence par une initialisation aléatoire du vecteur de poids de connexion $w_0^{(0)}, w_1^{(0)}, \ldots, w_p^{(0)}$, par exemple, $\vec{w} = \vec{0}$.

Puis, à chaque observation, on ajuste ce vecteur dans la direction opposée au gradient du risque empirique. En effet, ce gradient indique la direction de plus forte pente du risque empirique ; le redescendre nous rapproche d'un point où ce risque est minimal. Formellement, à une itération de l'algorithme, on tire une nouvelle observation $(\vec{x}^{\,i}, y^i)$ et on actualise, pour tout j, les poids de connexion de la façon suivante :

$$w_j \leftarrow w_j - \eta \frac{\partial L(f(\vec{x}^{\,i}), y^i)}{\partial w_j}. \tag{7.4}$$

Il est possible (et même recommandé dans le cas où les données ne sont pas extrêmement volumineuses) d'itérer plusieurs fois sur l'intégralité du jeu de données. Typiquement, on itère jusqu'à ce que l'algorithme converge à ϵ près.

Vitesse d'apprentissage

Cet algorithme a un hyperparamètre, $\eta > 0$, qui est le pas de l'algorithme du gradient et que l'on appelle la *vitesse d'apprentissage* (ou *learning rate*) dans le contexte des réseaux de neurones artificiels. Cet hyperparamètre joue un rôle important : s'il est trop grand, l'algorithme risque d'osciller autour de la solution optimale, voire de diverger. À l'inverse, s'il est trop faible, l'algorithme va converger très lentement. Il est donc essentiel de bien choisir sa vitesse d'apprentissage.

En pratique, on utilise souvent une vitesse d'apprentissage adaptative : relativement grande au début, puis de plus en plus faible au fur et à mesure que l'on se rapproche de la solution. Cette approche est à rapprocher d'algorithmes similaires développés dans le cas général de l'algorithme du gradient (comme par exemple la recherche linéaire par rebroussement, voir section A.3.4).

Classification binaire

Le cas de la classification binaire, utilisant un seuil comme fonction d'activation, est historiquement le premier à avoir été traité. La fonction de coût utilisée est connue sous le nom de *critère du perceptron* :

$$L(f(\vec{x}^{\,i}), y^i) = \max(0, -y^i o(\vec{x}^{\,i})) = \max(0, -y^i \langle \vec{w}, \vec{x} \rangle) \tag{7.5}$$

Ce critère est proche de la fonction d'erreur hinge. Quand la combinaison linéaire des entrées a le bon signe, le critère du perceptron est nul. Quand elle a le mauvais signe, le critère du perceptron est d'autant plus grand que cette combinaison linéaire est éloignée de 0. En utilisant ce critère, la règle d'actualisation 7.4 devient :

$$w_j \leftarrow \begin{cases} 0 & \text{si } y^i o(\vec{x}^{\,i}) > 0 \\ -y^i x_j^i & \text{sinon.} \end{cases}$$

Ainsi, quand le perceptron fait une erreur de prédiction, il déplace la frontière de décision de sorte à corriger cette erreur.

Théorème 7.1 (Théorème de convergence du perceptron) *Étant donné un jeu de n observations étiquetées $\mathcal{D} = \{(\vec{x}^{\,i}, y^i)_{i=1,\dots,n}\}$ et $D, \gamma \in \mathbb{R}_+^*$. Si :*
- $\forall i = 1, \dots, n,\ \left|\left|\vec{x}^{\,i}\right|\right|_2 \leq D$
- *il existe $\vec{u} \in \mathbb{R}^{p+1}$ tel que*
  - $||\vec{u}||_2 = 1$ *et*
  - $\forall i = 1, \dots, n,\ y^i \langle \vec{u}, \vec{x} \rangle \geq \gamma$

alors l'algorithme du perceptron converge en au plus $\left(\frac{D}{\gamma}\right)^2$ étapes. Ce théorème est un résultat de Albert Novikoff (1962). ■

Régression

Dans le cas de la régression, on utilise pour le coût empirique la fonction de coût quadratique :

$$L(f(\vec{x}^{\,i}), y^i) = \frac{1}{2}(y^i - f(\vec{x}^{\,i}))^2 = \frac{1}{2}\left(y^i - \langle \vec{w}, \vec{x} \rangle\right)^2. \tag{7.6}$$

La règle d'actualisation 7.4 devient :

$$w_j \leftarrow w_j - \eta(f(\vec{x}^{\,i}) - y^i)x_j^i. \tag{7.7}$$

Classification probabiliste

Dans le cas de la classification (binaire ou multi-classe), on utilise l'entropie croisée comme fonction de coût :

$$L(f(\vec{x}^{\,i}), y^i) = -\sum_{c=1}^{C} \delta(y^i, c) \log f_c(\vec{x}^{\,i}) = -\sum_{c=1}^{C} \delta(y^i, c) \log \frac{\exp(\langle \vec{w}^{\,c}, \vec{x} \rangle}{\sum_{k=1}^{C} \exp(\langle \vec{w}^{\,k}, \vec{x} \rangle)}.$$

Quelques lignes de calcul montrent que l'on obtient alors la même règle d'actualisation que pour la régression (7.7) :

$$w_j^k \leftarrow w_j^k - \eta(y_k^i - f(\vec{x}^{\,i}))x_j^i.$$

Ces calculs sont aussi valables dans le cas de la classification binaire, et la règle est exactement celle de la formule 7.7.

7.1.3 Modélisation de fonctions booléennes

Les perceptrons, étant capables d'apprendre des fonctions linéaires, sont donc capables d'exécuter certaines fonctions logiques sur des vecteurs d'entrées binaires. Ainsi, un perceptron dont les poids sont $w_0 = -1.5, w_1 = 1$ et $w_2 = 1$ calcule l'opérateur $\wedge$ (dont la table de vérité est rappelée sur la figure 7.3) et entre x_1 et x_2.

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

FIGURE 7.3 – Table de vérité du connecteur logique « et » ($\wedge$).

Par contraste, aucun perceptron ne peut apprendre la fonction « ou exclusif » (XOR), dont la table de vérité est rappelée sur la figure 7.4 : en effet, aucune droite ne peut séparer les points $(0, 0)$ et $(1, 1)$ d'un côté et $(0, 1)$ et $(1, 0)$ de l'autre.

x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

FIGURE 7.4 – Table de vérité du connecteur logique XOR.

7.2 Perceptron multi-couche

La capacité de modélisation du perceptron est limitée car il s'agit d'un modèle linéaire. Après l'enthousiasme généré par les premiers modèles connexionistes, cette réalisation a été à l'origine d'un certain désenchantement au début des années 1970.

Dans leur ouvrage de 1969 sur le sujet, Seymour Papert (mathématicien et éducateur) et Marvin Minksy (spécialiste en sciences cognitives) évoquent même leur intuition de l'inutilité d'étendre le perceptron à des architectures multi-couche : « *L'étude du perceptron s'est révélée fort intéressante en dépit de ses fortes limitations, et peut-être même grâce à elles. Plusieurs de ses caractéristiques attirent l'attention : sa linéarité, son intriguant théorème d'apprentissage, et la simplicité de son paradigme de calcul parallèle. Il n'y a aucune raison de supposer qu'aucune de ses vertus soient transposables à la version multi-couche. Néanmoins, nous considérons qu'il s'agit d'un problème de recherche important que d'expliciter (ou de rejeter) notre jugement intuitif de la stérilité d'une extension multi-couche.* » (Minsky and Papert, 1972)

Dans cette section, nous verrons comment l'histoire leur a donné tort.

7.2.1 Architecture

On appelle *perceptron multi-couche*, ou *multi-layer perceptron* (*MLP*) en anglais, un réseau de neurones construit en insérant des *couches intermédiaires* entre la couche d'entrée et celle de sortie d'un perceptron. On parlera parfois de *couches cachées* par référence à l'anglais *hidden layers*. Chaque neurone d'une couche intermédiaire ou de la couche de sortie reçoit en entrée les sorties des neurones de la couche précédente. Il n'y a pas de retour d'une couche vers une couche qui la précède ; on parle ainsi aussi d'un réseau de neurones *à propagation avant*, ou *feed-forward* en anglais.

En utilisant des fonctions d'activation non linéaires, telles que la fonction logistique ou la fonction tangente hyperbolique, on crée ainsi un modèle paramétrique hautement non linéaire.

Exemple

Prenons l'exemple d'un perceptron avec deux couches intermédiaires comme illustré sur la figure 7.5. Notons w_{jq}^h le poids de la connexion du neurone j de la couche $h-1$ au neurone q de la couche h, a_h la fonction d'activation utilisée en sortie de la couche h, et p_h le nombre de neurones dans la couche h.

La sortie z_q^1 du q-ème neurone de la première couche cachée vaut

$$z_q^1 = a_1\left(\sum_{j=0}^{p} w_{jq}^1 x_j\right).$$

La sortie z_q^2 du q-ème neurone de la deuxième couche cachée vaut

$$z_q^2 = a_2\left(\sum_{j=1}^{p_1} w_{jq}^2 z_j^1\right).$$

Enfin, la sortie du perceptron vaut

$$f(\vec{x}) = a_3\left(\sum_{j=1}^{p_2} w_j^3 z_j^2\right).$$

Ainsi, en supposant qu'on utilise une fonction logistique pour tous les neurones des couches cachées, la sortie du perceptron vaut

$$f(\vec{x}) = a_3\left(\sum_{j=0}^{p_2} w_{jq}^3 \frac{1}{1+\exp\left(-\sum_{j=0}^{p_1} w_{jq}^2 \frac{1}{1+\exp\left(-\sum_{j=0}^{p} w_{jq}^1 x_j\right)}\right)}\right),$$

ce qui devrait vous convaincre de la capacité du perceptron multi-couche à modéliser des fonctions non linéaires.

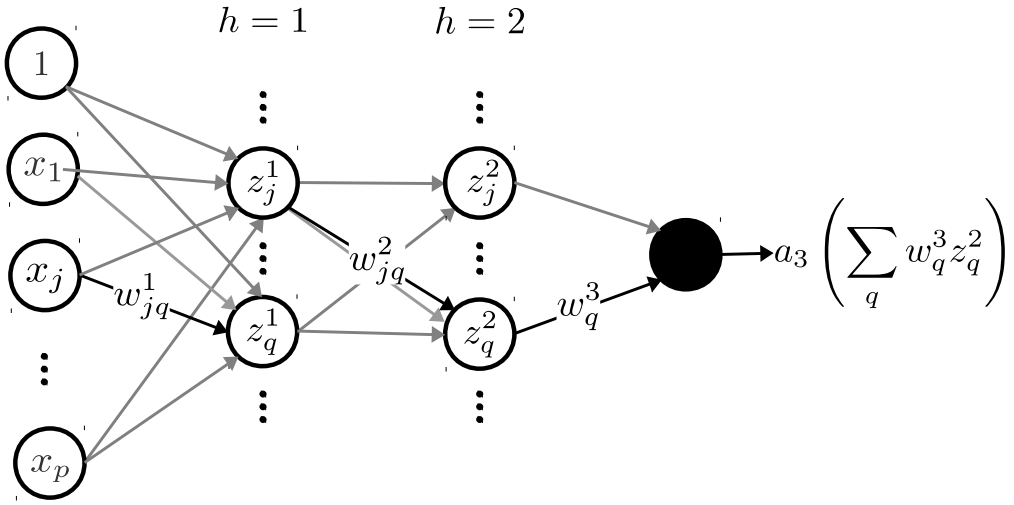


FIGURE 7.5 – Architecture d'un perceptron multi-couche.

Attention

Le perceptron multi-couche est un modèle paramétrique dont les paramètres sont les poids de connexions w_{jq}^h. (Le nombre de couches et leurs nombres de neurones font partie des hyperparamètres : on les suppose fixés, ce ne sont pas eux que l'on apprend). Ce modèle a donc d'autant plus de paramètres (c'est-à-dire de poids de connexion) qu'il y a de couches intermédiaires et de neurones dans ces couches. C'est pour éviter le sur-apprentissage que les réseaux de neurones profonds requièrent souvent des quantités massives de données pour apprendre de bons modèles.

7.2.2 Approximation universelle

Théorème 7.2 (Approximation universelle) *Soit $a : \mathbb{R} \to \mathbb{R}$ une fonction non constante, bornée, continue et croissante et K un sous-ensemble compact de $\mathbb{R}^P$. Étant donné $\epsilon > 0$ et une fonction f continue sur K, il existe un*

entier m, m scalaires $\{d_i\}_{i=1,\dots,m}$, m scalaires $\{b_i\}_{i=1,\dots,m}$, et m vecteurs $\{\vec{w}_i\}_{i=1,\dots,m}$ de $\mathbb{R}^p$ tels que pour tout $\vec{x} \in K$,

$$|f(\vec{x}) - \sum_{i=1}^{m} d_i \; a\left(\langle \vec{w}_i, \vec{x} \rangle + b_i\right)| < \epsilon.$$

En d'autres termes, toute fonction continue sur un sous-ensemble compact de $\mathbb{R}^p$ peut être approchée avec un degré de précision arbitraire par un perceptron multi-couche à une couche intermédiaire contenant un nombre fini de neurones.
■

Ce théorème, dû à George Cybenko (1989) et affiné par Kurt Hornik (1991), contraste fortement avec l'intuition de Papert et Minsky : nous pouvons maintenant apprendre toutes sortes de fonctions ! Attention cependant, ce résultat ne nous donne ni le nombre de neurones qui doivent composer cette couche intermédiaire, ni les poids de connexion à utiliser. Les réseaux de neurones à une seule couche cachée sont généralement peu efficaces, et on aura souvent de meilleurs résultats en pratique avec plus de couches.

7.2.3 Modéliser XOR avec un perceptron multi-couche

La fonction XOR, dont la table de vérité est donnée sur la figure 7.4, peut être facilement modélisée par un perceptron à une couche intermédiaire utilisant uniquement des fonctions de seuil comme fonctions d'activation. On peut par exemple utiliser une couche intermédiaire à 3 neurones, dont un biais, avec les poids suivants :

- $w^1_{01} = -0.5, w^1_{11} = 1, w^1_{21} = -1$, de sorte que $o^1_1 = -0.5 + x_1 - x_2$, ce qui permet de séparer $(1, 0)$ des autres possibilités ;
- $w^1_{02} = -0.5, w^1_{12} = -1, w^1_{22} = +1$, de sorte que $o^1_2 = -0.5 - x_1 + x_2$, ce qui permet de séparer $(0, 1)$ des autres possibilités ;
- $w^2_0 = -0.5, w^2_1 = 1, w^2_2 = 1$, de sorte que $f((x_1, x_2)) = -0.5 + \text{sign}(o^1_1) + \text{sign}(z^1_2)$ pour finir.

7.2.4 Entraînement par rétropropagation

Conditionnement et instabilité du gradient

Il est important de remarquer que la minimisation du risque empirique pour un perceptron multi-couche n'est *pas* un problème d'optimisation convexe. Ainsi, nous n'avons pas d'autres choix que d'utiliser un algorithme à directions de descente (voir section A.3.3), sans aucune garantie de converger vers un minimum global.

En effet, le gradient de la fonction de coût par rapport à un poids de connexion est généralement *mal conditionné*, ce qui signifie qu'une petite perturbation des conditions initiales de l'algorithme conduira à des résultats très différents. De plus, le gradient par rapport à un poids dans une des premières couches intermédiaires est souvent instable. Il aura tendance à être soit très faible, ce qui ralentira l'apprentissage (c'est ce que l'on appelle le problème de la *disparition du gradient*, ou *vanishing gradient* en anglais), soit à prendre des valeurs de plus en plus élevées à chaque itération, conduisant à l'inverse à un problème d'*explosion du gradient* (*exploding gradient* en anglais.)

L'initialisation des poids de connexion, la standardisation des variables, le choix de la vitesse d'apprentissage et celui des fonctions d'activation ont tous un impact sur la capacité du perceptron multi-couche à converger vers une bonne solution.

C'est pour cela que l'entraînement d'un réseau de neurones à plusieurs couches est délicat. Il a ainsi fallu attendre 2006 pour que les travaux de Geoff Hinton et d'autres, ainsi que l'amélioration de la puissance de calcul des ordinateurs, permettent de commencer à surmonter cette difficulté, remettant les réseaux de neurones sur le devant de la scène.

Saturation

Un autre problème du perceptron multi-couche apparaît quand la plupart des unités logistiques retournent des valeurs soit très proches de 0 soit très proches de 1. Cela arrive quand la somme pondérée o_j^h des signaux qu'ils reçoivent en entrée a une amplitude trop grande, autrement dit quand les poids de connexion ont eux-mêmes une amplitude trop élevée. Dans ces conditions, les actualisations de ces poids n'auront plus qu'un impact marginal sur les prédictions. On dit alors que le réseau est *saturé*.

Pour éviter cette situation, on peut se tourner vers la régularisation ℓ_2 (cf. section 6.2), appelée *dégradation des pondérations* ou *weight decay* dans le contexte des réseaux de neurones artificiels. Il s'agira d'ajouter au risque empirique $L(f(\vec{x}^{\,i}), y^i)$ la norme euclidienne du vecteur de poids de connexion.

Rétropropagation

Néanmoins, le principe fondamental de l'apprentissage d'un perceptron multi-couche, connu sous le nom de *rétropropagation* ou *backpropagation* (souvent raccourci en *backprop*), est connu depuis des décennies. Il repose, comme pour le perceptron sans couche intermédiaire, sur l'utilisation de l'algorithme du gradient pour minimiser, à chaque nouvelle observation, le risque $L(f(\vec{x}^{\,i}), y^i)$.

Pour actualiser le poids de connexion w_{jq}^h du neurone j de la couche $h-1$ vers le neurone q de la couche h, nous devons donc calculer $\frac{\partial L(f(\vec{x}^{\,i}, y^i))}{\partial w_{jq}^h}$. Pour ce faire, nous pouvons appliquer le théorème de dérivation des fonctions composées (*chain rule* en anglais). Nous notons o_j^h la combinaison linéaire des entrées du j-ème neurone de la couche h ; en d'autres termes, $z_j^h = a_h(o_j^h)$. Par convention, nous considérerons que $z_j^0 = x_j$. Ainsi,

$$\begin{aligned}
\frac{\partial L(f(\vec{x}^{\,i}), y^i)}{\partial w_{jq}^h} &= \frac{\partial L(f(\vec{x}^{\,i}), y^i)}{\partial o_q^h} \frac{\partial o_q^h}{\partial w_{jq}^h} = \frac{\partial L(f(\vec{x}^{\,i}), y^i)}{\partial z_q^h} \frac{\partial z_q^h}{\partial o_q^h} \frac{\partial o_q^h}{\partial w_{jq}^h} \\
&= \left(\sum_{r=1}^{p_{h+1}} \frac{\partial L(f(\vec{x}^{\,i}), y^i)}{\partial o_r^{h+1}} \frac{\partial o_r^{h+1}}{\partial z_q^h} \right) \frac{\partial z_q^h}{\partial o_q^h} \frac{\partial o_q^h}{\partial w_{jq}^h} \\
&= \left(\sum_{r=1}^{p_{h+1}} \frac{\partial L(f(\vec{x}^{\,i}), y^i)}{\partial o_r^{h+1}} w_{qr}^{h+1} \right) a_h'(o_q^h)\ z_j^{h-1}.
\end{aligned} \tag{7.8}$$

Ainsi, le gradient nécessaire à l'actualisation des poids de la couche h se calcule en fonction des gradients $\frac{\partial L(f(\vec{x}^{\,i}, y^i))}{\partial o_r^{h+1}}$ nécessaires pour actualiser les poids de la couche $(h+1)$.

Cela va nous permettre de simplifier nos calculs en utilisant une technique de *mémoïsation*, c'est-à-dire en évitant de recalculer des termes qui reviennent plusieurs fois dans notre procédure.

Plus précisément, l'entraînement d'un perceptron multi-couche par *rétropropagation* consiste à alterner, pour chaque observation $(\vec{x}^{\,i}, y^i)$ traitée, une phase de *propagation avant* qui permet de calculer les sorties de chaque neurone, et une phase de *rétropropagation des erreurs* dans laquelle on actualise les poids en partant de ceux allant de la dernière couche intermédiaire vers l'unité de sortie et en « remontant »le réseau vers les poids allant de l'entrée vers la première couche intermédiaire.

Exemple

Reprenons le réseau à deux couches intermédiaires décrit sur la figure 7.5, en utilisant l'identité comme dernière fonction d'activation a_3, une fonction d'erreur quadratique, et des activations logistiques pour a_1 et a_2. Nous rappelons que la dérivée de la fonction logistique peut s'écrire $\sigma'(u) = u'\sigma(u)(1-\sigma(u))$.

Lors de la propagation avant, nous allons effectuer les calculs suivants :

$$o_q^1 = \sum_{j=0}^{p} w_{jq}^1 x_j \ ; \ z_q^1 = \sigma(o_q^1)$$
$$o_q^2 = \sum_{j=1}^{p_1} w_{jq}^2 z_j^1 \ ; \ z_q^2 = \sigma(o_q^2)$$
$$o^3 = \sum_{j=1}^{p_2} w_j^3 z_j^2 \ ; \ f(\vec{x}^{\,i}) = z^3 = o^3.$$

Lors de la rétropropagation, nous calculons tout d'abord

$$\frac{\partial L(f(\vec{x}^{\,i}), y^i)}{\partial w_j^3} = \left(f(\vec{x}^{\,i}) - y^i\right) \frac{\partial f(\vec{x}^{\,i})}{\partial w_j^3} = \left(f(\vec{x}^{\,i}) - y^i\right) z_j^2$$

en utilisant les valeurs de $f(\vec{x}^{\,i})$ et z_j^2 que nous avons mémorisées lors de la propagation avant. Ainsi

$$w_j^3 \leftarrow w_j^3 - \eta \left(f(\vec{x}^{\,i}) - y^i\right) z_j^2.$$

Nous pouvons ensuite appliquer 7.8 et calculer

$$\frac{\partial L(f(\vec{x}^{\,i}), y^i)}{\partial w_{jq}^2} = \frac{\partial L(f(\vec{x}^{\,i}), y^i)}{\partial o_q^2} \frac{\partial o_q^2}{\partial w_{jq}^2}$$

où

$$\frac{\partial L(f(\vec{x}^{\,i}), y^i)}{\partial o_q^2} = \frac{\partial L(f(\vec{x}^{\,i}), y^i)}{\partial f(\vec{x}^{\,i})} w_q^3 \sigma'(o_q^2) = \left(f(\vec{x}^{\,i}) - y^i\right) w_q^3 z_q^2 (1 - z_q^2) \tag{7.9}$$

et

$$\frac{\partial o_q^2}{\partial w_{jq}^2} = z_j^1.$$

Nous pouvons donc utiliser les valeurs de $f(\vec{x}^{\,i})$, z_q^2 et z_j^1 mémorisées lors de la propagation avant, et w_q^3 que nous venons d'actualiser, pour actualiser w_{jq}^2 par

$$w_{jq}^2 \leftarrow w_{jq}^2 - \eta \left(f(\vec{x}^{\,i}) - y^i\right) w_q^3 z_q^2 (1 - z_q^2) \, z_j^1.$$

Enfin, nous pouvons de nouveau appliquer 7.8 et calculer

$$\frac{\partial L(f(\vec{x}^{\,i}), y^i)}{\partial w_{jq}^1} = \left(\sum_{r=1}^{p_2} \frac{\partial L(f(\vec{x}^{\,i}), y^i)}{\partial o_r^2} w_{qr}^2 \right) z_q^1 (1 - z_q^1) \, x_j.$$

Encore une fois, nous disposons de tous les éléments nécessaires : z_q^1 a été calculé lors de la propagation avant, les poids w_{qr}^2 ont été actualisés à l'étape précédente, et les dérivées partielles $\frac{\partial L(f(\vec{x}^{\,i}), y^i)}{\partial o_q^2}$ ont elles aussi été calculées à l'étape précédente (7.9). Nous pouvons donc effectuer aisément notre dernière étape de rétropropagation :

$$w_{jq}^1 \leftarrow w_{jq}^1 - \eta \left(\sum_{r=1}^{p_2} \frac{\partial L(f(\vec{x}^{\,i}), y^i)}{\partial o_r^2} w_{qr}^2 \right) z_q^1 (1 - z_q^1) \, x_j.$$

Remarque

Il est possible d'ajouter une unité de biais à chaque couche intermédiaire ; les dérivations se font alors sur le même principe.

7.2.5 Et le deep learning dans tout ça ?

De l'analyse d'images à la voiture autonome en passant par la reconnaissance vocale, la plupart des avancées récentes mises en avant par la presse grand public dans le domaine de l'apprentissage automatique reposent sur les *réseaux de neurones profonds*, ou *deep neural nets*. Rentrer en détail dans ce domaine dépasse le cadre de cet ouvrage introductif, ainsi nous n'en abordons ici que les grandes lignes.

Le domaine de l'apprentissage profond repose fondamentalement sur les principes que nous venons de voir. En effet, un perceptron multi-couche est profond dès lors qu'il contient suffisamment de couches – la définition de « suffisamment » dépendant des auteurs. Le domaine de l'apprentissage profond s'intéresse aussi à de nombreuses autres architectures comme les *réseaux récurrents* (*RNN*, pour *Recursive Neural Nets*), et en particulier *Long Short-Term Memory (LSTM) networks* pour modéliser des données séquentielles (telles que du texte ou des données temporelles) et les *réseaux convolutionnels* (*CNN*, pour *Convolutional Neural Nets*) ou les *réseaux de capsules* (*CapsNets*) pour le traitement d'images. Dans tous les cas, il s'agit essentiellement

— d'utiliser ces architectures pour créer des modèles paramétriques (potentiellement très complexes) ;
— d'en apprendre les poids par un algorithme à directions de descente.

Les défis du deep learning

L'apprentissage des poids de connexion d'un réseau de neurones profond pose cependant des difficultés techniques : en effet, le problème d'optimisation à résoudre n'est pas convexe, et converger vers un « bon » minimum local n'est pas une tâche facile. Cette tâche est d'autant plus difficile que le réseau est complexe, et les progrès dans ce domaine ne sont possibles que grâce au développement de méthodes pour la rendre plus aisée.

De plus, les réseaux de neurones profonds ont de nombreux paramètres, et requièrent donc l'utilisation de grands volumes de données pour éviter le sur-apprentissage. Il est donc généralement nécessaire de les déployer sur des architectures distribuées.

Ainsi, malgré son succès dans certains domaines d'application, le deep learning est délicat à mettre en place et n'est pas toujours la meilleure solution pour résoudre un problème de machine learning, en particulier face à un petit jeu de données.

L'apprentissage de représentations

On associe souvent aux réseaux de neurones profond la notion de *representation learning*. En effet, il est possible de considérer que chacune des couches intermédiaires successives apprend une nouvelle représentation des données $(z_1^h, z_2^h, \dots, z_{p_h}^h)$ à partir de la représentation de la couche précédente, et ce jusqu'à pouvoir appliquer entre la dernière couche intermédiaire et la sortie du réseau un algorithme linéaire. Cet aspect est exploité en particulier dans les *auto-encodeurs* que nous verrons à la section 11.3.3

Points clefs

— Le perceptron permet d'apprendre des modèles paramétriques linéaires et est entraîné par des actualisations itératives de ses poids grâce à un algorithme à directions de descente.

— Le perceptron multi-couche permet d'apprendre des modèles paramétriques non linéaires et offre une grande flexibilité de modélisation.

— Le perceptron multi-couche est entraîné par rétropropagation, qui combine le théorème de dérivation des fonctions composées avec une mémoïsation pour une grande efficacité de calcul.

— Le problème d'optimisation du perceptron multi-couche et, plus généralement, de tout réseau de neurones artificiel profond, n'est pas convexe, et il n'est pas facile d'obtenir un bon minimum.

Pour aller plus loin

- Nous n'avons dans ce chapitre qu'esquissé les briques de bases du deep learning. Pour aller plus loin, nous recommandons la lecture de l'ouvrage de Goodfellow et al. (2016).
- `http://playground.tensorflow.org/` permet de jouer avec l'architecture et l'entraînement d'un réseau de neurones profond.

Bibliographie

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4) :303–314.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT Press, Cambridge, MA. `http://www.deeplearningbook.org/`.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2) :251–257.

Minsky, M. and Papert, S. (1972). *Perceptrons : an Introduction to Computational Geometry*. MIT Press, Cambridge, MA.

Novikoff, A. B. J. (1962). On convergence proofs on perceptrons. In *Symposium on the Mathematical Theory of Automata*, pages 615–622.

Rosenblatt, F. (1957). The perceptron–a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory.

Chapitre 8

Méthode des plus proches voisins

Ce chapitre présente un algorithme de prédiction non paramétrique conceptuellement très simple mais néanmoins puissant, qui permet de construire des frontières de décision complexes sans faire aucune hypothèse sur la distribution des données. Cet algorithme, dit des plus proches voisins, se base sur le principe de « qui se ressemble s'assemble », et utilise les étiquettes des exemples les plus proches pour prendre une décision.

La puissance de cet algorithme venant du choix de la fonction utilisée pour définir cette proximité entre observations, ce chapitre sera aussi l'occasion de définir des distances et similarités pour différentes représentations des données. Dans ce chapitre, nous étudierons aussi des cas où la représentation par un vecteur réel de dimension p n'est pas nécessairement la plus appropriée.

Enfin, nous montrerons comment les plus proches voisins peuvent être utilisés en pratique dans le cadre du filtrage collaboratif.

Objectifs

- Implémenter l'algorithme des k plus proches voisins
- Calculer distances et similarités pour différents types de représentations des données
- Définir la frontière de décision de l'algorithme du plus proches voisins
- Expliquer pourquoi l'algorithme des k plus proches voisins est susceptible de ne pas marcher en haute dimension.

8.1 Méthode du plus proche voisin

8.1.1 Méthode

Définition 8.1 (Algorithme du plus proche voisin) Étant donné un jeu $\mathcal{D} = \{(\vec{x}^{\,i}, y^i)_{i=1,\dots,n}\}$ de n observations étiquetées, et une distance d sur $\mathcal{X}$, on appelle *algorithme du plus proche voisin* l'algorithme consistant à étiqueter une nouvelle observation $\vec{x}$ par l'étiquette du point du jeu d'entraînement qui en est la plus proche :

$$f(\vec{x}) = y^{\arg\min_{i=1,\dots,n} d(\vec{x},\vec{x}^{\,i})}.$$

■

Cet algorithme s'applique aussi bien à un problème de classification qu'à un problème de régression.

8.1.2 Diagramme de Voronoï

L'algorithme du plus proche voisin crée une partition de $\mathcal{X}$ en n parties : la i-ème de ces parties est l'ensemble des points de $\mathcal{X}$ dont le plus proche voisin dans $\mathcal{D}$ est $\vec{x}^{\,i}$.

$$\mathcal{X} = \bigcup_{i=1}^{n} \{\vec{x} \in \mathcal{X} \ : \ d(\vec{x}, \vec{x}^{\,i}) \leq d(\vec{x}, \vec{x}^{\,l}) \ \ \forall \vec{x}^{\,l} \in \mathcal{D}\}$$

Cette partition est ce que l'on appelle un *diagramme de Voronoï.*

Définition 8.2 (Diagramme de Voronoï) Soient un espace métrique $\mathcal{X}$, muni de la distance $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, et un ensemble fini $\mathcal{S}$ d'éléments de $\mathcal{X}$.

Étant donné un élément $\vec{u} \in \mathcal{S}$ (appelé *germe*), on appelle *cellule de Voronoï* l'ensemble défini par

$$\text{Vor}(\vec{u}) = \{\vec{x} \in \mathcal{X} : \forall\ \vec{v} \in \mathcal{S},\ d(\vec{x}, \vec{u}) \leq d(\vec{x}, \vec{v})\}.$$

Une cellule de Voronoï est un ensemble convexe, et l'union de toutes les cellules de Voronoï forme une partition de $\mathcal{X}$ appelée *diagramme de Voronoï* :

$$\mathcal{X} = \bigcup_{\vec{u} \in \mathcal{S}} \text{Vor}(\vec{u}).$$

■

La figure 8.1 présente un exemple de diagramme de Voronoï, calculé dans le plan en utilisant la distance euclidienne. Les cellules sont des polygones convexes. Tous les points à l'intérieur d'une même cellule ont la même étiquette que leur graine.

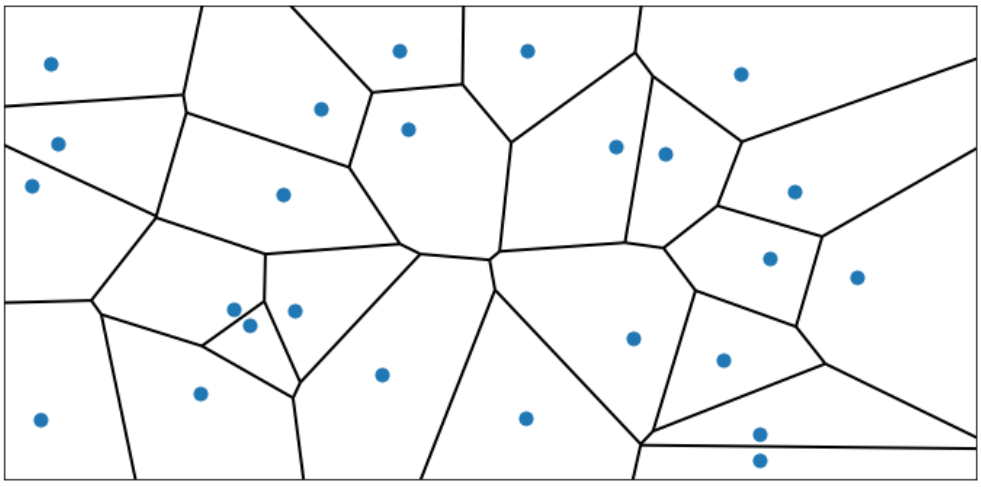


FIGURE 8.1 – Diagramme de Voronoï pour 25 points en deux dimensions, en utilisant la distance euclidienne.

Une autre façon de présenter l'algorithme du plus proche voisin est de le voir comme une partition de $\mathcal{X}$ en autant de parties qu'il existe d'étiquettes distinctes dans le jeu d'entraînement. La partie correspondant à l'étiquette y est l'union des cellules de Voronoï correspondant à une observation de $\mathcal{D}$ étiquetée par y.

$$\mathcal{X} = \bigcup_{y} \{\vec{x} \in \mathcal{X} : y^{\arg\min_{i=1,\ldots,n} d(\vec{x}, \vec{x}^{\,i})} = y\}$$

Ainsi, cet algorithme très simple permet de construire une frontière de décision beaucoup plus complexe qu'un algorithme paramétrique linéaire, comme on le voit sur la figure 8.2.

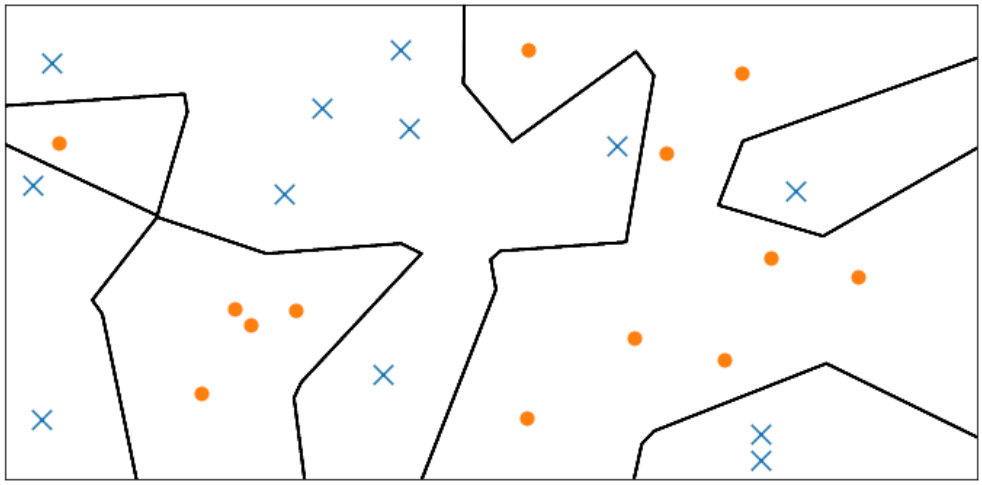


FIGURE 8.2 – Frontière de décision d'un algorithme du plus proche voisin pour les données de la figure 8.1, maintenant étiquetées.

8.2 Méthode des plus proches voisins

La méthode du plus proche voisin a le défaut d'être très sensible au bruit : si une observation est mal étiquetée, ou (ce qui est assez probable en raison d'une part du bruit de mesure et d'autre part de la nature vraisemblablement incomplète de notre représentation) mal positionnée, tous les points dans sa cellule de Voronoï seront mal étiquetés. Pour rendre cette méthode plus robuste, on se propose de combiner les « opinions » de *plusieurs* voisins de l'observation que l'on cherche à étiqueter.

8.2.1 Méthode des k plus proches voisins

Définition 8.3 (Algorithme des k plus proches voisins) Étant donné un jeu $\mathcal{D} = \{(\vec{x}^{\,i}, y^i)_{i=1,\dots,n}\}$ de n observations étiquetées, une distance d sur $\mathcal{X}$, et un hyperparamètre $k \in \mathbb{N}^*$, on appelle *algorithme des k plus proches voisins*, ou *kNN* pour *k nearest neighbors* en anglais, l'algorithme consistant à étiqueter une nouvelle observation $\vec{x}$ en fonction des étiquettes des k points du jeu d'entraînement dont elle est la plus proche. En notant $\mathcal{N}_k(\vec{x})$ l'ensemble des k plus proches voisins de $\vec{x}$ dans $\mathcal{D}$:

— Pour un problème de classification, on applique le *vote de la majorité*, et $\vec{x}$ prend l'étiquette majoritaire parmi celles de ses k plus proches voisins :

$$f(\vec{x}) = \arg\max_c \sum_{i:\vec{x}^{\,i} \in \mathcal{N}_k(\vec{x})} \delta(y^i, c).$$

— Pour un problème de régression, $\vec{x}$ prend comme étiquette la *moyenne* des étiquettes de ses k plus proches voisins :

$$f(\vec{x}) = \frac{1}{k} \sum_{i:\vec{x}^{\,i} \in \mathcal{N}_k(\vec{x})} y^i.$$

■

Dans le cas ou $k = 1$, on retrouve l'algorithme du plus proche voisin.

L'algorithme des k plus proches voisins est un exemple d'apprentissage *non paramétrique* : la fonction de décision s'exprime en fonction des données observées et non pas comme une formule analytique fonction des variables.

On peut rapprocher son fonctionnement de celui d'un *raisonnement par cas*, qui consiste à agir en se remémorant les choix déjà effectuées dans des situations semblables précédemment rencontrées, qui se retrouve par exemple lorsqu'un médecin traite un patient en se remémorant comment d'autres patients avec des symptômes similaires ont guéri.

La frontière de décision de l'algorithme des k plus proches voisins est linéaire par morceaux ; quand k augmente, elle devient plus « simple », car le vote de la majorité permet de lisser les aspérités créées par les exemples individuels, comme on le voit sur la figure 8.3.

FIGURE 8.3 – Frontière de décision d'un algorithme des 5 plus proches voisins pour les données des figures 8.1 et 8.2.

8.2.2 Apprentissage paresseux

On parle parfois d'*apprentissage paresseux*, ou *lazy learning* en anglais, pour qualifier l'algorithme des k plus proches voisins. En effet, la procédure d'apprentissage consiste uniquement à *stocker* les données du jeu d'entraînement et ne comporte aucun calcul. Attention, cela peut être un facteur limitant, au niveau de la mémoire, si le jeu d'entraînement est très grand.

À l'inverse, la procédure de prédiction requiert de calculer la distance de l'observation à étiqueter à *chaque* observation du jeu d'entraînement, ce qui peut être intensif en temps de calcul si ce jeu d'entraînement est très grand. Une prédiction requiert en effet de calculer n distances, une opération d'une complexité de l'ordre de $\mathcal{O}(np)$ en p dimensions, puis de trouver les k plus petites de ces distances, une opération d'une complexité en $\mathcal{O}(n \log k)$.

8.2.3 Nombre de plus proches voisins

Comme nous l'avons décrit ci-dessus, nous proposons d'utiliser $k > 1$ pour rendre l'algorithme des k plus proches voisins plus robuste au bruit. Cependant, à l'inverse, si $k = n$, l'algorithme prédira, dans le cas d'un problème de classification, la classe majoritaire dans $\mathcal{D}$, et dans le cas d'un problème de régression, la moyenne des étiquettes de $\mathcal{D}$, ce qui paraît tout aussi peu satisfaisant.

Il faudra donc choisir une valeur de k intermédiaire, ce que l'on fera généralement en utilisant une validation croisée. L'heuristique $k \approx \sqrt{n}$ est aussi parfois utilisée.

8.2.4 Variantes

ϵ-voisins

Plutôt que de considérer un nombre fixe de voisins les plus proches, on peut préférer considérer tous les exemples d'apprentissage suffisamment proches de l'observation à étiqueter : cela permet de mieux utiliser le jeu d'entraînement dans les zones où il est dense. De plus, les prédictions faites en se basant sur des exemples proches (non pas relativement, mais dans l'absolu) sont intuitivement plus fiables que celles faites en se basant sur des exemples éloignés.

Définition 8.4 (Algorithme des ϵ-voisins) Étant donné un jeu $\mathcal{D} = \{(\vec{x}^{\,i}, y^i)_{i=1,\dots,n}\}$ de n observations étiquetées, une distance d sur $\mathcal{X}$, et un hyperparamètre $\epsilon \in \mathbb{R}_+$, on appelle *algorithme des ϵ-voisins*, ou *ϵ-ball neighbors* en anglais, l'algorithme consistant à étiqueter une nouvelle observation $\vec{x}$ en fonction des étiquettes de tous les points du jeu d'entraînement situés à une distance inférieure à ϵ de $\vec{x}$.

— Pour un problème de classification :

$$f(\vec{x}) = \arg\max_c \sum_{i:d(\vec{x},\vec{x}^{\,i})\leq\epsilon} \delta(y^i, c).$$

— Pour un problème de régression :

$$f(\vec{x}) = \frac{1}{|\{i : d(\vec{x}, \vec{x}^{\,i}) \leq \epsilon\}|} \sum_{i:d(\vec{x},\vec{x}^{\,i})\leq\epsilon} y^i.$$

■

Une limitation évidente de cet algorithme est qu'il est nécessaire de définir une stratégie alternative en l'absence d'exemples dans la boule de rayon ϵ choisi.

Pondération des voisins

Alternativement, pour prendre en compte la notion que les voisins véritablement proches sont plus fiables pour la prédiction que ceux plus éloignés, on peut *pondérer* la contribution de chacun des voisins en fonction de sa distance à l'observation à étiqueter, typiquement par

$$w_i = \frac{1}{d(\vec{x}, \vec{x}^{\,i})} \text{ ou } w_i = e^{-\left(\frac{1}{2}d(\vec{x},\vec{x}^{\,i})\right)}.$$

Dans le cas d'un problème de classification, on compare, pour chaque classe c, la somme des contributions de chaque voisin appartenant à cette classe :

$$f(\vec{x}) = \arg\max_c \sum_{i:x^i\in\mathcal{N}_k(\vec{x})} \delta(y^i, c) w_i.$$

Dans le cas d'un problème de régression, il s'agit simplement de pondérer la moyenne :

$$f(\vec{x}) = \frac{1}{k} \sum_{\vec{x}^{\,i}\in\mathcal{N}_k(\vec{x})} w_i y^i.$$

8.3 Distances et similarités

L'ingrédient essentiel de l'algorithme des k plus proches voisin est la *distance* permettant de déterminer quelles sont les observations du jeu d'entraînement les plus proches du point à étiqueter.

Attention

L'algorithme des k plus proches voisins est sensible aux attributs non pertinents ; en effet, ceux-ci seront pris en compte dans le calcul de la distance et pourront la biaiser.

De plus, en haute dimension, cet algorithme souffre du fléau de la dimension (cf. 11.1.3) : tous les exemples seront loin de l'observation que l'on cherche à étiqueter, et l'intuition selon laquelle on peut utiliser les étiquettes des exemples proches pour faire une prédiction ne fonctionnera plus.

8.3.1 Distances

Rappelons ici la définition d'une distance :

Définition 8.5 (Distance) Étant donné un ensemble $\mathcal{X}$, on appelle *distance* sur $\mathcal{X}$ toute fonction $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ vérifiant les trois propriétés suivantes :

1. Séparation : $\forall \vec{u}, \vec{v} \in \mathcal{X} \times \mathcal{X},\ d(\vec{u}, \vec{v}) = 0 \Leftrightarrow \vec{u} = \vec{v}$
2. Symétrie : $\forall \vec{u}, \vec{v} \in \mathcal{X} \times \mathcal{X},\ d(\vec{u}, \vec{v}) = d(\vec{v}, \vec{u})$
3. Inégalité triangulaire : $\forall \vec{t}, \vec{u}, \vec{v} \in \mathcal{X}^3,\ d(\vec{u}, \vec{v}) \leq d(\vec{u}, \vec{t}) + d(\vec{t}, \vec{v})$.

■

Dans le cas où $\mathcal{X} = \mathbb{R}^p$, on utilisera le plus souvent une distance de Minkowski :

Définition 8.6 (Distance de Minkowski) Étant donné q fixé, $q \geq 1$, on appelle *distance de Minkowski* la distance définie par

$$d_q : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$$
$$\vec{u}, \vec{v} \mapsto ||\vec{u} - \vec{v}||_q = \left(\sum_{j=1}^{p} |u_j - v_j|^q \right)^{1/q}.$$

La *distance euclidienne* est le cas particulier $q = 2$: $d_2(\vec{u}, \vec{v}) = \sqrt{\sum_{j=1}^{p} (u_j - v_j)^2}$.

Quand $q = 1$, on parle de *distance de Manhattan*, ou *distance du chauffeur de taxi*, car dans le plan elle consiste à calculer la distance parcourue entre $\vec{u}$ et $\vec{v}$ en se déplaçant uniquement parallèlement aux axes, comme ce serait le cas d'un taxi dans Manhattan, où les rues forment un quadrillage : $d_1(\vec{u}, \vec{v}) = \sum_{j=1}^{p} |u_j - v_j|$.

Enfin, la *distance* ℓ^∞, ainsi nommée en référence à l'espace de suites ℓ^∞, et notée d_∞, est la différence maximale entre $\vec{u}$ et $\vec{v}$ sur une dimension :

$$d_\infty(\vec{u}, \vec{v}) = \max_{j=1,\ldots,p} |u_j - v_j|.$$

On l'appelle aussi *distance de Tchebychev*. ■

Théorème 8.1 *La distance ℓ_∞ est la limite quand $q \to \infty$ de la distance de Minkowski.* ■

Démonstration. Posons $k = \arg\max_{j=1,\dots,p} |u_j - v_j|$. Alors
— soit $|u_k - v_k| = 0$, et donc $|u_j - v_j| = 0\ \forall j$, auquel cas la distance de Minkowski entre $\vec{u}$ et $\vec{v}$ vaut 0 quel que soit q et donc $\lim_{q\to\infty} d_q(\vec{u},\vec{v}) = 0 = d_\infty(\vec{u},\vec{v})$;
— soit $|u_k - v_k| > 0$, et on peut écrire

$$d_q(\vec{u},\vec{v}) = |u_k - v_k| \left(\sum_{j=1}^{p} \left(\frac{|u_j - v_j|}{|u_k - v_k|} \right)^q \right)^{1/q} \leq |u_j - v_j| p^{1/q}$$

car pour tout j, $|u_j - v_j| \leq |u_k - v_k|$, avec égalité au maximum p fois. D'autre part,

$$d_q(\vec{u},\vec{v}) = \left(|u_k - v_k|^q + \sum_{j\neq k} |u_j - v_j|^q \right)^{1/q} \geq |u_k - v_k|$$

puisque $|u_j - v_j|^q > 0$ pour tout j. Ainsi

$$|u_k - v_k| \leq d_q(\vec{u},\vec{v}) \leq |u_j - v_j| p^{1/q}.$$

Comme $\lim_{q\to\infty} p^{1/q} = 1$, $\lim_{q\to\infty} d_q(\vec{u},\vec{v}) = |u_k - v_k|$.

□

8.3.2 Similarités entre vecteurs réels

Cependant, il n'est pas nécessaire pour appliquer l'algorithme des k plus proches voisins d'utiliser une distance : une notion de *similarité* est suffisante.

Définition 8.7 (Similarité) On appelle *similarité* sur $\mathcal{X}$ toute fonction $s : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}_+$ qui est d'autant plus grande que les deux éléments de $\mathcal{X}$ qu'elle compare sont semblables. ■

Contrairement à une distance, une similarité n'a pas de propriétés mathématiques particulières. Il est possible de transformer n'importe quelle distance en similarité, en utilisant par exemple la transformation $s(\vec{u},\vec{v}) = -d(\vec{u},\vec{v})$, ou $s(\vec{u},\vec{v}) = \frac{1}{1+d(\vec{u},\vec{v})}$.
Une notion de similarité fréquemment utilisée lorsque $\mathcal{X} = \mathbb{R}^p$ est celle du coefficient de corrélation.

Définition 8.8 (Coefficient de corrélation) Étant donnés $\vec{u}, \vec{v} \in \mathbb{R}^p$, on appelle *coefficient de corrélation* ou *corrélation de Pearson* entre $\vec{u}$ et $\vec{v}$ la valeur

$$\rho(\vec{u},\vec{v}) = \frac{\sum_{j=1}^{p} \left(u_j - \frac{1}{p}\sum_{k=1}^{p} u_k \right) \left(v_j - \frac{1}{p}\sum_{k=1}^{p} v_k \right)}{\sqrt{\sum_{j=1}^{p} \left(u_j - \frac{1}{p}\sum_{k=1}^{p} u_k \right)^2} \sqrt{\sum_{j=1}^{p} \left(v_j - \frac{1}{p}\sum_{k=1}^{p} v_k \right)^2}}.$$

■

Remarquons que si l'on appelle $\vec{u'}$ (resp. $\vec{v'}$) la version centrée de $\vec{u}$ (resp. $\vec{v}$), c'est-à-dire le vecteur obtenu en lui retranchant la moyenne de ses coordonnées : $\vec{u'} = \vec{u} - \frac{1}{p}\sum_{j=1}^{p} u_j$, ce coefficient se simplifie en

$$\rho(\vec{u},\vec{v}) = \frac{\sum_{j=1}^{p} u'_j v'_j}{\sqrt{\sum_{j=1}^{p} {u'_j}^2} \sqrt{\sum_{j=1}^{p} {v'_j}^2}} = \frac{\langle \vec{u'}, \vec{v'} \rangle}{||\vec{u'}||_2 \, ||\vec{v'}||_2}$$

et vaut donc le cosinus de l'angle entre $\vec{u}'$ et $\vec{v}'$. C'est pour cette raison qu'on appelle parfois ρ la *similarité cosinus*.

Si en plus d'être centrés, les vecteurs $\vec{u}$ et $\vec{v}$ sont normalisés, la similarité cosinus se réduit au produit scalaire entre $\vec{u}$ et $\vec{v}$. La valeur absolue de ce produit scalaire est d'autant plus grande que les vecteurs $\vec{u}$ et $\vec{v}$ sont colinéaires, et vaut 0 quand ces vecteurs sont orthogonaux.

Nous verrons au chapitre 10 comment étendre la notion de produit scalaire sur $\mathbb{R}^p$ à celle de *noyau* sur $\mathcal{X}$, ce qui nous permettra de définir aisément d'autres mesures de similarité.

Remarque

En général, la similarité cosinus et la distance euclidienne ne définissent pas les mêmes voisinages.

8.3.3 Similarités entre ensembles

Jusqu'à présent dans cet ouvrage, nous avons uniquement traité de données qui peuvent être représentées par un vecteur réel. Cependant, il arrive souvent que nos observations soient plus aisément représentées comme des sous-ensembles d'un même ensemble fini d'éléments. Par exemple, une chaîne de caractères peut être représentée par l'ensemble des lettres qu'elle contient, ou un document texte par l'ensemble des mots qu'il contient.

Ces représentations peuvent être transformées en vecteurs binaires : un sous-ensemble $\mathcal{S}$ de $\mathcal{E}$ peut être représenté comme un vecteur binaire de taille $|\mathcal{E}|$ dont chaque bit correspond à un élément e de $\mathcal{E}$ et vaut 1 si $e \in \mathcal{S}$ et 0 sinon. On peut alors appliquer les distances ou similarités précédentes à ces représentations vectorielles.

Cependant, on peut aussi définir des distances ou des similarités directement sur ces ensembles, sans passer par une représentation vectorielle dont la dimension pourrait être très grande (autant que le nombre de mots dans le dictionnaire, dans le cas de la représentation d'un document par les mots qu'il contient), bien que potentiellement parcimonieuse (la plupart des documents contiendront peu de mots, relativement au dictionnaire).

On peut choisir de comparer deux ensembles en fonction du nombre d'éléments qui apparaissent dans un seul d'entre eux.

Définition 8.9 (Distance de Hamming) La *distance de Hamming* entre deux sous-ensembles $\mathcal{S}$ et $\mathcal{T}$ de $\mathcal{E}$ est définie comme

$$H(\mathcal{S}, \mathcal{T}) = |\mathcal{S} \triangle \mathcal{T}|,$$

où $\mathcal{S} \triangle \mathcal{T} = \{e \in \mathcal{S} \setminus \mathcal{T}\} \cup \{e \in \mathcal{T} \setminus \mathcal{S}\}$ est la différence symétrique entre $\mathcal{S}$ et $\mathcal{T}$.

En utilisant une représentation binaire de $\mathcal{S}$ et $\mathcal{T}$, la distance de Hamming est le nombre de bits différents entre ces deux vecteurs, et est équivalente à leur distance de Manhattan. ■

Deux ensembles sont considérés être d'autant plus semblables qu'ils ont d'éléments en commun. Attention, si l'on compare des ensembles susceptibles d'être de tailles très différentes, cela doit être comparé au nombre d'éléments qu'ils *pourraient* avoir en commun : deux ensembles de grande taille auront naturellement plus d'éléments communs que deux ensembles de petite taille. On utilisera pour cela la similarité de Jaccard.

Définition 8.10 (Similarité de Jaccard/Tanimoto) Étant donné un ensemble $\mathcal{E}$ d'éléments, on appelle *similarité de Jaccard*, *similarité de Tanimoto*, ou encore *index de Jaccard* la fonction

$$\begin{aligned} J : 2^{\mathcal{E}} \times 2^{\mathcal{E}} &\to [0, 1] \\ \mathcal{S}, \mathcal{T} &\mapsto \frac{|\mathcal{S} \cap \mathcal{T}|}{|\mathcal{S} \cup \mathcal{T}|}. \end{aligned}$$

Ici $2^{\mathcal{E}}$ désigne l'ensemble des parties de $\mathcal{E}$, autrement dit l'ensemble de ses sous-ensembles. Cette notation est choisie car, du moins dans le cas fini, un sous-ensemble $\mathcal{S} \subseteq \mathcal{E}$ peut être représenté comme une fonction binaire de $\mathcal{E}$ vers $\{0, 1\}$ qui associe 1 à un élément de $\mathcal{E}$ présent dans $\mathcal{S}$, et 0 aux autres. ■

Si les éléments sont susceptibles d'apparaître plusieurs fois dans chaque « ensemble » (il ne s'agira alors plus d'ensembles, mais de multi-ensembles), on peut prendre en compte la multiplicité des éléments en utilisant la similarité MinMax :

Définition 8.11 (Similarité MinMax) Étant donné un multi-ensemble $\mathcal{S}$ de $\mathcal{E}$, et un élément $e \in \mathcal{E}$, on note $m_{\mathcal{S}}(e)$ la multiplicité de e dans $\mathcal{S}$, autrement dit son nombre d'occurrences dans $\mathcal{S}$. Étant donné un ensemble $\mathcal{E}$ d'éléments, on appelle *similarité MinMax* entre deux multi-ensembles $\mathcal{S}$ et $\mathcal{T}$ de $\mathcal{E}$ la fonction qui retourne

$$\mathrm{MinMax}(\mathcal{S}, \mathcal{T}) = \frac{\sum_{e \in \mathcal{S} \cap \mathcal{T}} \min(m_{\mathcal{S}}(e), m_{\mathcal{T}}(e))}{\sum_{e \in \mathcal{S} \cup \mathcal{T}} \max(m_{\mathcal{S}}(e), m_{\mathcal{T}}(e))}.$$

■

8.3.4 Similarités entre données catégoriques

Un autre type de variable que l'on peut fréquemment rencontrer est celui des variables *catégoriques*, comme le genre ou la catégorie socio-professionnelle d'un individu, le jour de la semaine d'un événement, ou le stade d'une tumeur. Ces variables sont représentées par un élément d'une liste de possibilités plutôt que par un nombre.

Bien que l'on puisse arbitrairement associer un nombre à chacune de ces possibilités, et représenter la variable par ce nombre, ce n'est généralement pas une bonne idée. En effet, deux catégories représentées par des nombres voisins seront considérées par l'algorithme d'apprentissage comme plus proches que deux catégories représentées par des nombres plus éloignées.

Il n'existe pas toujours de distance très claire entre deux catégories : la profession « cadre de la fonction publique » est-elle plus proche de « artisan » ou de « employé de commerce » ? Il nous est surtout possible, a priori, de distinguer deux cas : soit deux observations appartiennent à la même catégorie, soit elles appartiennent à deux catégories distinctes. Pour pouvoir appliquer une méthode des plus proches voisins avec une distance définie par une norme ℓ_q, ou une méthode paramétrique, à une telle variable, on utilisera souvent un encodage *one-hot*.

Définition 8.12 (Encodage one-hot) Étant donnée une variable catégorique pouvant prendre m valeurs distinctes, on appelle *encodage one-hot* la représentation de cette variable par un vecteur binaire de dimension m, dans lequel chaque bit correspond à une des valeurs, et est mis à 1 si la variable prend cette valeur et à 0 sinon. ■

Exemple

Supposons que nos observations soient des articles de journaux, et que nous disposions d'une variable catégorique identifiant leur sujet comme un parmi « politique », « société », « économie », « sport », « science » ou « culture ». L'encodage one-hot de cette variable consistera à la remplacer par 6 variables binaires, valant $(1, 0, 0, 0, 0, 0)$ si l'article parle de politique et $(0, 0, 0, 0, 0, 1)$ s'il parle de science.

Une fois les données ainsi encodées, on peut leur appliquer les distances et similarités définies sur les vecteurs réels.

Remarque

Dans le cas des jours de la semaine (ou des mois de l'année), le problème de l'encodage par un nombre de 1 à 7 (ou de 1 à 12) est la nature cyclique de ces catégories : février (2) est certes plus proche – temporellement – de mai (5-2=3) que de juillet (7-2=5), mais il est encore plus proche de décembre (12-2=10). Pour conserver cette proximité temporelle, on pourra positionner ces valeurs de manière équidistante sur un cercle de rayon 1, et représenter chacune d'entre elles par deux variables, le cosinus et le sinus de cette position. On représentera ainsi février par $(\cos(\frac{2\pi}{6}), \sin(\frac{2\pi}{6})) = (\frac{1}{2}, \frac{\sqrt{3}}{2})$, mai par $(\cos(\frac{5\pi}{6}), \sin(\frac{5\pi}{6})) = (-\frac{\sqrt{3}}{2}, \frac{1}{2})$, et décembre par $(\cos(2\pi), \sin(2\pi)) = (1, 0)$. La distance euclidienne de février à mai est de $\sqrt{2}$ tandis que celle de février à décembre est de 1.

8.4 Filtrage collaboratif

Le *filtrage collaboratif*, ou *collaborative filtering* en anglais, est à la base des systèmes de recommandation, tels que ceux utilisés par Netflix ou Amazon, qui utilisent les évaluations d'un groupe pour proposer des recommandations à un individu.

Il existe de nombreuses méthodes pour résoudre ce problème. Néanmoins, l'une d'entre elles repose sur les principes que nous venons de voir dans ce chapitre, en faisant l'hypothèse que des utilisateurs aux goûts similaires vont continuer à aimer les mêmes choses.

Plus précisément, étant donnés un ensemble $\mathcal{S}$ d'objets (films, livres, achats, etc.), un ensemble $\mathcal{X}$ d'utilisateurs, nous supposons l'existence d'une fonction $r : \mathcal{X} \times \mathcal{S} \mapsto \mathbb{R}$ telle que $r(u, a)$ est la note donnée par l'utilisateur u à l'objet a. Cette fonction est partiellement connue, au sens où tous les utilisateurs n'ont pas noté tous les objets. En notant, pour tout $(a, b) \in \mathcal{S} \times \mathcal{S}$, $\mathcal{U}(a, b)$ le sous-ensemble de $\mathcal{X}$ contenant uniquement les utilisateurs qui ont noté à la fois a et b, et pour tout $u \in \mathcal{X}$, $\bar{r}(u)$ la note moyenne donnée par u, on peut alors définir une similarité entre objets sur la base de la similarité cosinus :

$$s(a, b) = \frac{\sum_{u \in \mathcal{U}(a,b)} (r(u, a) - \bar{r}(u)) (r(u, b) - \bar{r}(u))}{\sqrt{\sum_{u \in \mathcal{U}(a,b)} (r(u, a) - \bar{r}(u))^2 \sum_{u \in \mathcal{U}(a,b)} (r(u, b) - \bar{r}(u))^2}}. \tag{8.1}$$

Appelons maintenant $\mathcal{N}_u^k(a)$ les k plus proches voisins de a parmi les objets notés par u. On peut recommander l'objet a pour l'utilisateur u par :

$$f(u, a) = \frac{\sum_{b \in \mathcal{N}_u^k(a)} s(a, b) r(u, b)}{\sum_{b \in \mathcal{N}_u^k(a)} |s(a, b)|}.$$

Points clefs

- L'algorithme des k plus proches voisins a un entraînement paresseux ; pour compenser, le coût algorithmique d'une prédiction peut être élevé si la base de données d'entraînement est grande.
- La qualité des prédictions d'un algorithme des k plus proches voisins dépend principalement du choix d'une bonne distance ou similarité.
- L'algorithme des k plus proches voisins fonctionne mieux en *faible dimension* :
  - son exécution est plus rapide ;
  - il est moins susceptible d'être biaisé par des variables qui ne sont pas pertinentes ;
  - il est moins susceptible de souffrir du fléau de la dimension.

Pour aller plus loin

- L'ouvrage de Webb (1999) présente un éventail de distances qui peuvent être utilisées dans le cadre de l'algorithme des k plus proches voisins.
- Des structures de données telles que les *arbres kd* (Bentley, 1957) ou l'utilisation de *tables de hachage* permettent de stocker le jeu d'entraînement de sorte à retrouver plus facilement les plus proches voisins. L'idée sous-jacente de ces structures est de regrouper les exemples qui sont proches les uns des autres.
- Des ouvrages entiers ont été écrits sur le sujet des systèmes de recommandation. On pourra notamment se référer à ceux de Ricci et al. (2016) ou Aggarwal (2016).
- Un ensemble d'articles sur les k plus proches voisins a été rassemblé dans Aha (1997).
- L'article de Hechenbichler et Schliep (2006) fournit une bonne revue des méthodes de pondération des plus proches voisins.

Bibliographie

Aggarwal, C. C. (2016). *Recommender Systems*. Springer.

Aha, D. W. (1997). Special issue on lazy learning. *Artificial Intelligence Review*, 11(1–5) :7–423.

Bentley, J. L. (1957). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9) :509–517.

Hechenbichler, K. et Schliep, K. (2006). Weighted k-nearest-neighbor techniques and ordinal classification. In *SFB 386*. Discussion paper 399.

Ricci, F., Rokach, L., et Shapira, B. (2016). *Recommender Systems Handbook*. Springer, 2 edition.

Webb, A. (1999). *Statistical Pattern Recognition*. Arnold, London.

Chapitre 9

Arbres et forêts

L'algorithme des plus proches voisins permet de construire des modèles non paramétriques *métriques*, c'est-à-dire qu'ils reposent sur la définition d'une distance ou similarité pertinente entre les observations. Cela n'est pas toujours chose aisée, et les *arbres de décision* approchent le problème différemment. *Non métriques*, *hiérarchiques*, et non paramétriques, ils présentent d'intéressantes propriétés, en particulier en ce qui concerne les attributs discrets. Cependant, ils ont tendance à mal apprendre, et à avoir de faibles propriétés de généralisation.

Dans ce chapitre, après avoir exploré plus en détail les propriétés des arbres et leur construction, nous verrons comment les *combiner* pour en faire des modèles puissants, dits *modèles ensemblistes*, qui sont bien plus que la somme de leurs parties.

Objectifs

— Détailler les avantages et inconvénients d'un arbre de décision

— Construire un arbre de décision

— Savoir combiner des apprenants faibles parallèlement ou séquentiellement et en comprendre l'intérêt

9.1 Arbres de décision

9.1.1 Apprentissage hiérarchique

Les modèles que nous avons vus jusqu'à présent procèdent en une seule opération pour étiqueter une observation $\vec{x}$. De tels modèles utilisent le même ensemble de variables pour toutes les classes, et leur accordent la même importance pour toutes les observations. Ces modèles ne sont pas bien adaptés aux cas où les classes ont une distribution multi-modale, c'est-à-dire qu'elles sont déterminées par différentes variables dans différentes régions de l'espace, ou quand les variables sont discrètes.

Par contraste, les arbres de décision sont des *modèles hiérarchiques*, qui se comportent comme une série successive de tests conditionnels, dans laquelle chaque test dépend de ses antécédents. Ils sont couramment utilisés en dehors du monde du machine learning, par exemple pour décrire les étapes d'un diagnostic ou d'un choix de traitement pour un médecin, ou les chemins possibles dans un « livre dont vous êtes le héros ». La figure 9.1 présente un tel arbre de décision.

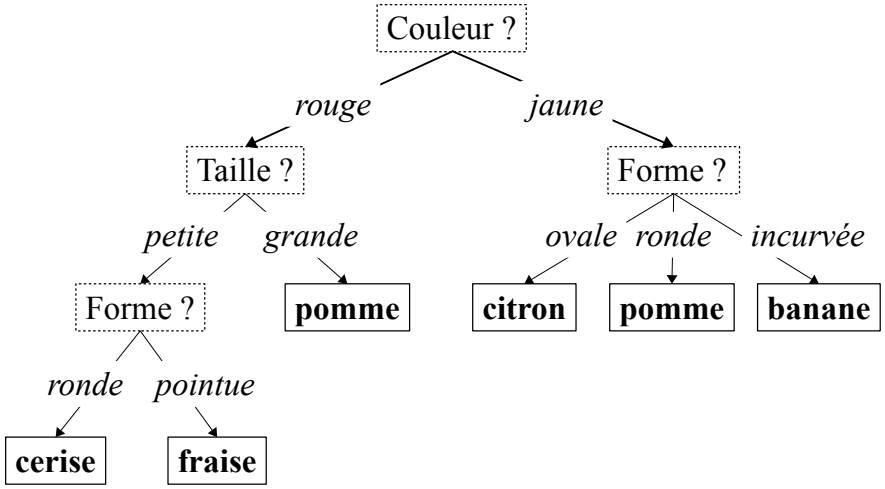


FIGURE 9.1 – Exemple d'arbre de décision pour étiqueter un fruit.

La figure 9.1 nous permet d'illustrer trois propriétés des arbres de décision :
— Ils permettent de traiter des attributs discrets (comme ici la forme, la taille et la couleur), sans nécessiter une notion de similarité ou d'ordre sur ces attributs (on parle d'apprentissage *non métrique*) ;
— Ils permettent de traiter un problème de classification multi-classe sans passer par des classifications binaires ;
— Ils permettent de traiter des classes multi-modales (comme ici pour l'étiquette « pomme », qui est affectée à un fruit grand et rouge ou à un fruit jaune et rond.)

Définition 9.1 (Arbre de décision) On appelle *arbre de décision* un modèle de prédiction qui peut être représenté sous la forme d'un arbre. Chaque nœud de l'arbre teste une condition sur une variable et chacun de ses enfants correspond à une réponse possible à cette condition. Les feuilles de l'arbre correspondent à une étiquette.

Pour prédire l'étiquette d'une observation, on « suit » les réponses aux tests depuis la racine de l'arbre, et on retourne l'étiquette de la feuille à laquelle on arrive. ■

9.1.2 Partition de l'espace par un arbre de décision

Un arbre de décision partitionne l'espace $\mathcal{X}$ des observations en autant de régions qu'il a de feuilles ; au sein d'une même région, toutes les observations reçoivent alors la même étiquette. Dans le cas d'un problème de classification, cette étiquette est l'étiquette majoritaire dans la région. En supposant n observations $\vec{x}^{\,1}, \vec{x}^{\,2}, \ldots, \vec{x}^{\,n}$ de $\mathcal{X}$ étiquetées par $y^1, y^2, \ldots, y^n$, et R régions $R_1, R_2, \ldots, R_R$, on peut écrire

$$f(\vec{x}) = \sum_{r=1}^{R} \delta_{\vec{x} \in R_r} \operatorname*{arg\,max}_{c=1,\ldots,C} \sum_{i:\vec{x}^{\,i} \in R_r} \delta(y^i, c). \tag{9.1}$$

Dans ce chapitre, nous ne traiterons pas séparément du cas binaire et du cas multi-classe car le premier découle du second en étiquetant les deux classes 1 et 2 plutôt que 0 et 1 comme nous en avons l'habitude.

Pour un problème de régression, cette étiquette est l'étiquette moyenne des observations dans cette région :

$$f(\vec{x}) = \sum_{r=1}^{R} \delta_{\vec{x} \in R_r} \frac{1}{|R_r|} \sum_{i:\vec{x}^{\,i} \in R_r} y^i. \tag{9.2}$$

Ces notations sont lourdes et on préférera éviter d'essayer d'écrire la fonction de décision d'un arbre de décision sous forme analytique. Cependant, il est important de retenir qu'un arbre de décision partitionne le jeu d'entraînement de manière récursive en des sous-ensembles de plus en plus petits. Une telle partition est illustrée sur la figure 9.2 pour des variables réelles.

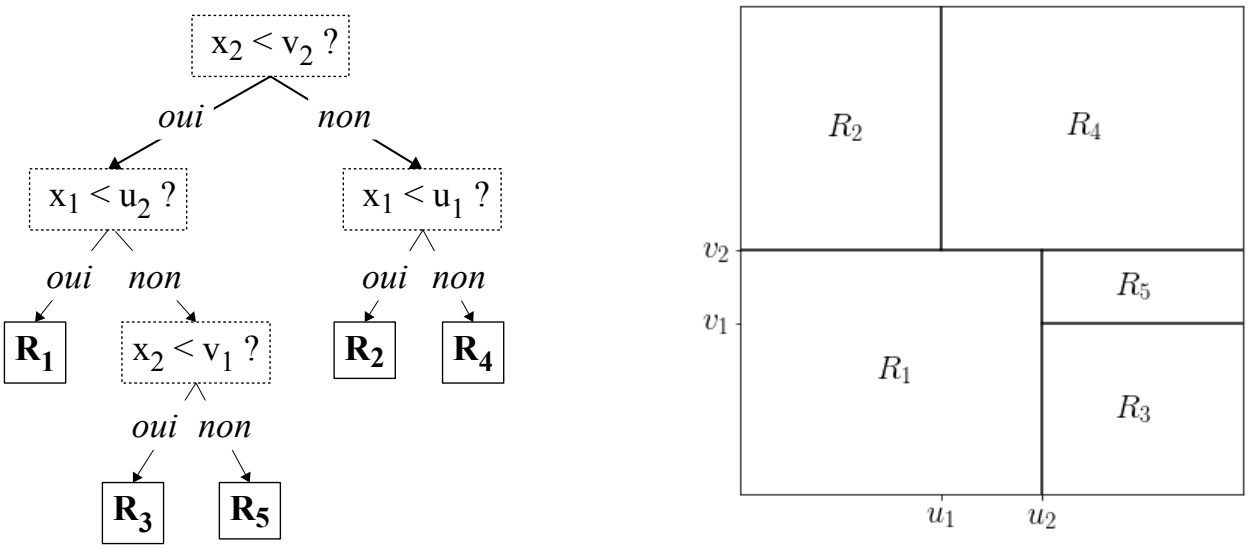


FIGURE 9.2 – L'arbre de décision (à gauche) partitionne $\mathbb{R}^2$ en 5 zones (à droite)

9.2 Comment faire pousser un arbre

9.2.1 CART

L'algorithme utilisé pour entraîner un arbre de décision est appelé *CART*, pour *Classification And Regression Tree* (Breiman et al., 1984). Il s'agit d'un algorithme de partitionnement de l'espace par une approche gloutonne, récursive et divisive.

CART peut être utilisé pour entraîner un arbre de décision *binaire*, c'est-à-dire dans lequel chaque nœud a exactement deux enfants, sans perte de généralité. En effet, tout arbre peut être ré-exprimé sous la forme d'un arbre binaire, comme l'illustre la figure 9.3.

Comme illustré sur la figure 9.2, CART partitionne les données *une variable à la fois*, ce qui produit une frontière de décision orthogonale aux axes.

Nous supposons par la suite que les données sont définies dans un espace $\mathcal{X}$ de dimension p.

Définition 9.2 (Variable séparatrice) À chaque nœud d'un arbre de décision construit par CART correspond une *variable séparatrice* (*splitting variable*) $j \in \{1, \dots, p\}$ selon laquelle vont être partitionnées les données. Cette variable séparatrice définit deux régions, correspondant aux enfants du nœud considéré.

Dans le cas où la variable de séparation est *binaire*, ces deux régions sont

$$R_l(j, s) = \{\vec{x} : x_j = 0\}; \qquad R_r(j, s) = \{\vec{x} : x_j = 1\}.$$

Dans le cas où la variable de séparation est une variable *discrète* pouvant prendre plus de deux valeurs (ou modalités), elle s'accompagne alors d'un sous-ensemble de ces valeurs $\subset \text{dom}(x_j)$ Les deux régions sont

$$R_l(j, s) = \{\vec{x} : x_j \in \}; \qquad R_r(j, s) = \{\vec{x} : x_j \notin \}.$$

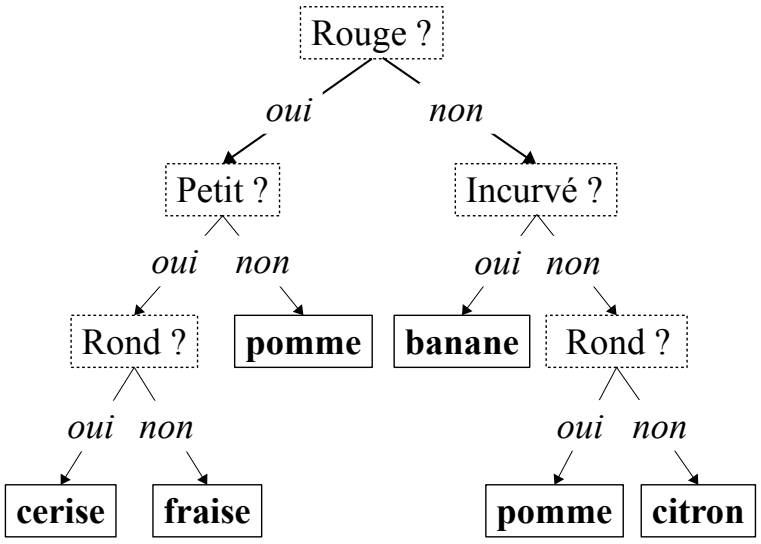


FIGURE 9.3 – Version binaire de l'arbre de décision de la figure 9.1.

Enfin, dans le cas où la variable de séparation est une variable *réelle*, elle s'accompagne alors d'un *point de séparation* (*splitting point*) s qui est la valeur de l'attribut par rapport à laquelle va se faire la décision. Les deux régions sont alors

$$R_l(j,s) = \{\vec{x} : x_j < s\}; \qquad R_r(j,s) = \{\vec{x} : x_j \geq s\}.$$

■

À chaque itération de l'algorithme CART, on itère sur toutes les valeurs possibles de j et, le cas échéant, toutes les valeurs possibles de s ou $\mathcal{S}$ pour déterminer le couple (j,s) qui minimise un *critère* prédéfini.

Remarque

Dans le cas d'une variable continue x_j, si l'on suppose les valeurs prises par cette variable dans $\mathcal{D}$ ordonnées : $x_j^1 \leq x_j^2 \leq \ldots, \leq x_j^n$, alors les valeurs possibles de s sont $\frac{x_j^{i+1} - x_j^i}{2}$ pour toutes les valeurs de i telles que $x_j^{i+1} \neq x_j^i$.

Dans le cas d'un problème de régression, ce critère est généralement l'erreur quadratique moyenne : à quel point la partition en (j,s) permet-elle de regrouper des observations d'étiquettes proches ? On choisit donc la variable et le point de séparation comme

$$\arg\min_{j,s} \left(\sum_{i:\vec{x}^i \in R_l(j,s)} (y^i - y_l(j,s))^2 + \sum_{i:\vec{x}^i \in R_r(j,s)} (y^i - y_r(j,s))^2 \right)$$

où $y_l(j,s)$ (resp. $y_r(j,s)$) est l'étiquette associée à la région $R_l(j,s)$ (resp. $R_r(j,s)$) à ce stade, soit donc la moyenne des étiquettes de cette région.

Dans le cas d'un problème de classification, on utilise plutôt que l'erreur quadratique moyenne un *critère d'impureté*, ainsi appelé car il quantifie à quel point la région considérée est « polluée » par des éléments des classes qui n'y sont pas majoritaires. En notant Imp ce critère, on choisit donc la variable et le point de séparation comme

$$\arg\min_{j,s} \left(\frac{|R_l(j,s)|}{n} \text{Imp}(R_l(j,s)) + \frac{|R_r(j,s)|}{n} \text{Imp}(R_r(j,s)). \right)$$

Encore une fois, il s'agit d'un algorithme glouton : il n'y a aucune garantie que cette stratégie aboutisse à l'arbre de décision dont l'impureté ou l'erreur quadratique moyenne est minimale.

9.2.2 Critères d'impureté

Il existe plusieurs critères d'impureté, que nous détaillons dans cette section : l'*erreur de classification*, l'*entropie croisée* et l'*impureté de Gini*. Pour les définir, nous allons utiliser la notation $p_c(R)$ pour indiquer la proportion d'exemples d'entraînement de la région R qui appartiennent à la classe c :

$$p_c(R) = \frac{1}{|R|} \sum_{i:\vec{x}^{\,i} \in R} \delta(y^i, c).$$

Tout d'abord, l'*erreur de classification* permet de définir l'impureté d'une région R comme la proportion d'exemples de cette région qui n'appartiennent pas à la classe majoritaire.

$$\text{Imp}(R) = 1 - \max_{c=1,\dots,C} p_c(R). \tag{9.3}$$

Si tous les exemples d'une région appartiennent à la même classe, l'erreur de classification de cette région vaut 0 ; à l'inverse, si une région contient autant d'exemples de chacune des C classes, $p_c(R) \approx \frac{1}{C}$ quelle que soit la classe c, et l'erreur de classification vaut $1 - \frac{1}{C}$, soit $\frac{1}{2}$ dans le cas d'une classification binaire.

Ensuite, l'*entropie croisée* permet de définir l'impureté d'une région R de sorte à choisir la séparation qui maximise le gain d'information : le but de la construction est alors de minimiser la quantité d'information supplémentaire nécessaire pour étiqueter correctement les exemples d'entraînement de R.

$$\text{Imp}(R) = -\sum_{c=1}^{C} p_c(R) \log_2 p_c(R). \tag{9.4}$$

Si tous les exemples d'une région appartiennent à la même classe, l'entropie croisée de cette région vaut 0 ; à l'inverse, si une région contient autant d'exemples de chacune des C classes, l'entropie croisée vaut $\log_2(C)$, soit 1 dans le cas d'une classification binaire.

Enfin, la définition la plus utilisée de l'impureté est l'*impureté de Gini*, qui permet de quantifier la probabilité qu'un exemple du jeu d'entraînement soit mal étiqueté s'il était étiqueté aléatoirement en fonction de la distribution des étiquettes dans R.

Définition 9.3 (Impureté de Gini) L'*impureté de Gini* d'une région R est définie comme

$$\text{Imp}(R) = \sum_{c=1}^{C} p_c(R) \left(1 - p_c(R)\right).$$

■

Si tous les exemples d'une région appartiennent à la même classe, l'impureté de Gini de cette région vaut 0 ; à l'inverse, si une région contient autant d'exemples de chacune des C classes, l'impureté de Gini vaut $1 - \frac{1}{C}$, soit $\frac{1}{2}$ dans le cas d'une classification binaire.

9.2.3 Élaguer un arbre

Nous avons établi une stratégie pour construire un arbre. Il nous faut maintenant pouvoir décider quand l'arrêter. En effet, si un arbre peu profond risque de mal modéliser le problème, un arbre trop profond est susceptible de sur-apprendre.

Une approche classique du problème est d'arrêter de diviser une région quand celle-ci contient un nombre minimum d'exemples d'entraînement fixé par avance. Cela évite de construire des feuilles trop spécifiques ne contenant qu'un seul point. On peut aussi choisir de limiter la profondeur de l'arbre.

Il est également possible d'utiliser une méthode de *régularisation* (cf. chapitre 6) pour contrôler la complexité de l'arbre, mesurée par le nombre de régions qu'il définit. C'est ce qu'on appelle l'*élagage par coût en complexité*, ou *cost-complexity pruning* : le coût en complexité d'un arbre T est donné par

$$C_\lambda(T) = \sum_{r=1}^{|T|} n_r \text{Imp}(R_r) + \lambda |T|, \tag{9.5}$$

où $|T|$ est le nombre de régions définies par T, R_r la r-ème de ces régions, et n_r le nombre d'exemples d'entraînement qu'elle contient. $\lambda > 0$ est un hyperparamètre permettant de contrôler l'importance relative de la mesure d'erreur et de la complexité. Cette procédure requiert de développer complètement l'arbre, puis de l'élaguer en regroupant séquentiellement certaines régions jusqu'à ce que le critère soit optimal.

Malheureusement, les arbres de décision ont tendance à donner des modèles trop simples et à avoir des performances de prédiction à peine supérieures à des modèles aléatoires et peu robustes aux variations dans les données. On les qualifie d'*apprenants faibles* (*weak learners* en anglais). Heureusement, il est possible d'y remédier grâce aux méthodes ensemblistes.

9.3 Méthodes ensemblistes : la sagesse des foules

Les méthodes ensemblistes sont des méthodes très puissantes en pratique, qui reposent sur l'idée que combiner de nombreux apprenants faibles permet d'obtenir une performance largement supérieure aux performances individuelles de ces apprenants faibles, car leurs erreurs se compensent les unes les autres.

Cette idée est similaire au concept de *sagesse des foules*, ou *wisdom of crowd* : si je demande à mes élèves l'année de la mort de Pompidou, il est probable que la moyenne de leurs réponses soit assez proche de la bonne (1974) ; cependant, si je demande cette date à une seule personne au hasard, je n'aurais aucun moyen de savoir a priori si cette personne connaît la date ou me répond au hasard.

Exemple

Pour illustrer ce concept, imaginons une tâche de classification en deux dimensions, dans laquelle les deux classes sont séparées par une diagonale, mais que le seul algorithme d'apprentissage dont nous disposions ne puisse apprendre qu'une frontière de décision en escalier, avec un nombre limité de paliers. Combiner des dizaines voire des centaines de ces frontières de décision en escalier peut nous donner une bien meilleure approximation de la véritable frontière de décision. Cet exemple est illustré sur la figure 9.4.

Attention

La théorie des méthodes ensemblistes montre que lorsque les modèles que l'on combine ont été appris par des *apprenants faibles*, c'est-à-dire simples à entraîner et peu performants, ces méthodes permettent d'améliorer la performance par rapport à celle du meilleur de ces modèles individuels. En pratique, si les modèles individuels sont déjà performants et robustes au bruit, le modèle ensembliste ne sera pas nécessairement meilleur. On utilise le plus souvent des arbres de décision comme modèles individuels.

Deux grandes familles d'approches permettent de créer *plusieurs* modèles faibles à partir d'un unique jeu de données. Elles sont toutes deux basées sur un ré-échantillonage du jeu de données, mais sont conceptuellement très différentes. La première, le *bagging*, est une méthode *parallèle* dans laquelle les apprenants

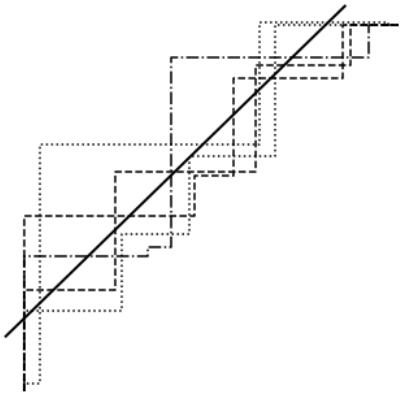


FIGURE 9.4 – Chacune des frontières de décision en escalier est une mauvaise approximation de la vraie frontière qui est la diagonale en trait plein. Cependant, combiner ces escalier permet une meilleure approximation de la diagonale.

faibles sont indépendants les uns des autres, tandis que la deuxième, le *boosting*, est une méthode *séquentielle* dans laquelle chaque nouvel apprenant est construit en fonction des performances du précédent.

9.3.1 Méthodes parallèles : le bagging

Supposons un jeu de données $\mathcal{D}$ de n observations $\vec{x}^{\,1}, \vec{x}^{\,2}, \ldots, \vec{x}^{\,n}$ de $\mathcal{X}$ étiquetées par $y^1, y^2, \ldots, y^n$. Le *bagging*, proposé par Leo Breiman (1996), consiste à former B versions de $\mathcal{D}$ par *échantillonage bootstrap* (cf. section 3.1.4). Chaque modèle faible est entraîné sur un des échantillons, ce qui peut être fait en parallèle. Les B prédictions sont ensuite combinées

— par vote de la majorité dans le cas d'un problème de classification ;
— en prenant la moyenne dans le cas d'un problème de régression.

Le bagging permet de réduire la variance des estimateurs individuels. C'est ce qui lui permet d'atteindre une plus grande stabilité et une meilleure prédiction qu'eux.

La figure 9.5 illustre ce principe : les 5 premiers arbres qui composent le classifieur entraîné par bagging séparent nettement moins bien le jeu de données que le bagging, mais ils font des erreurs sur des régions différentes de l'espace, qui se compensent lorsqu'on les combine.

Forêts aléatoires

La puissance des méthodes ensemblistes se révèle lorsque les apprenants faibles sont indépendants conditionnellement aux données, autrement dit aussi différents les uns des autres que possible, afin que leurs erreurs puissent se compenser les unes les autres. Pour atteindre cet objectif, l'idée des *forêts aléatoires*, proposée toujours par Leo Breiman, est de construire les arbres individuels non seulement sur des échantillons différents (comme pour le bagging), mais aussi en utilisant des *variables* différentes (Breiman, 2001).

Plus précisément, les arbres construits pour former une forêt aléatoire diffèrent de ceux appris par CART en ce que, à chaque nœud, on commence par sélectionner $q < p$ variables aléatoirement, avant de choisir la variable séparatrice *parmi celles-ci*. En classification, on utilise typiquement $q \approx \sqrt{p}$, ce qui permet aussi de réduire considérablement les temps de calculs puisqu'on ne considère que peu de variables

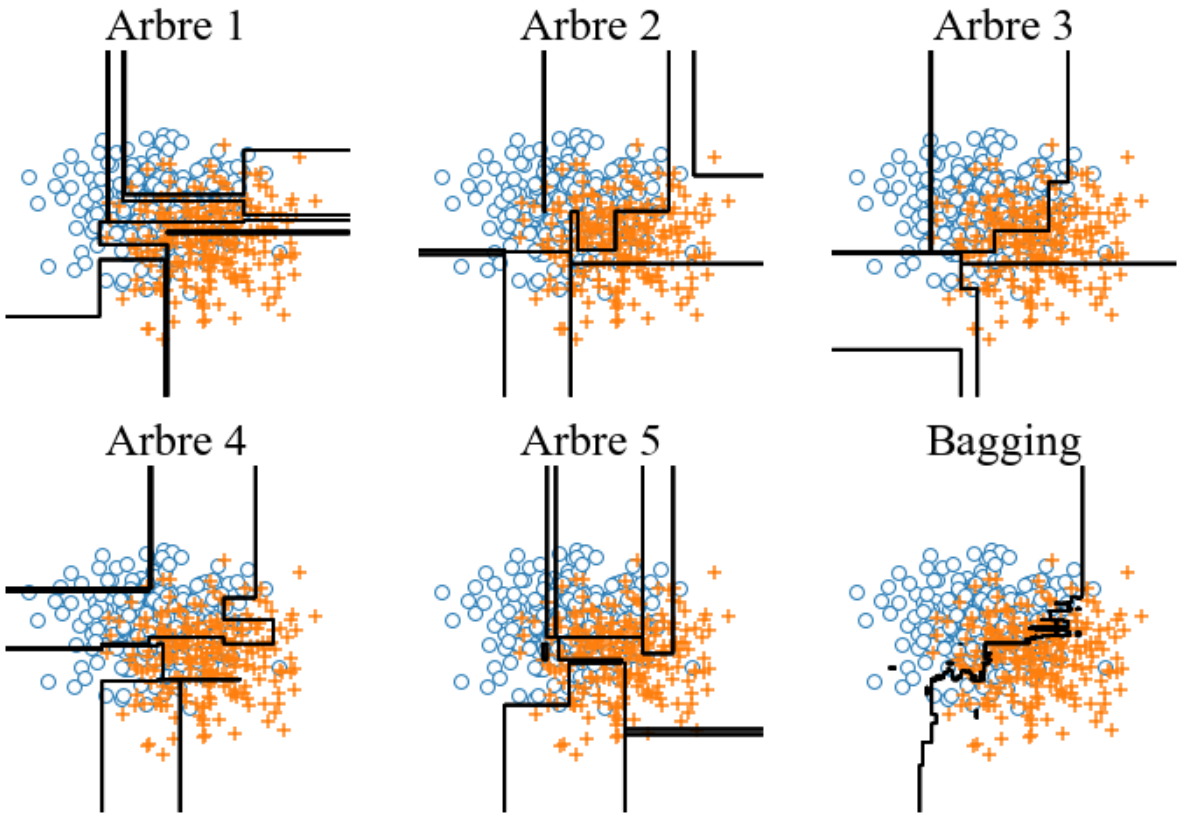


FIGURE 9.5 – Performance sur un jeu de test d'un classifieur entraîné par bagging (en bas à droite) et des 5 premiers arbres qui le composent.

à chaque nœud (5 pour un problème à 30 variables, 31 pour un problème avec 1000 variables). Pour la régression, le choix par défaut est plutôt de $q \approx \frac{p}{3}$.

Remarque

Le fait de moyenner les réponses des différents arbres permet d'utiliser les forêts aléatoires aussi bien pour des problèmes de régression que de classification.

En pratique, les forêts aléatoires sont un des algorithmes les plus performants et les plus simples à mettre en place. Elles ont aussi l'avantage de peu dépendre de leurs hyperparamètres, à savoir du nombre q de variables considérées à chaque nœud, du nombre d'observations utilisées pour chaque arbre (n dans la procédure que nous avons décrite, mais que l'on pourrait réduire), du nombre maximum d'observations dans les feuilles de l'arbre (généralement fixé à 1 en classification et 5 en régression), et du nombre d'arbres, à partir du moment où celui-ci est suffisamment grand.

9.3.2 Méthodes séquentielles : le boosting

Alors que le bagging repose sur l'hypothèse que des apprenants ayant de bonnes performances sur des régions différentes de l'espace $\mathcal{X}$ des observations auront des erreurs décorrélées, et qu'il est donc possible de faire confiance à la majorité d'entre eux pour ne pas faire d'erreur pour une prédiction donnée,

l'approche *séquentielle* de la construction d'un ensemble cherche à créer des modèles faibles qui se concentreront sur les erreurs du modèle. On parle alors de *boosting* : par itérations successives, des apprenants faibles viennent exalter (« booster ») les performances du modèle final qui les combine.

La première proposition dans ce sens est l'algorithme AdaBoost, proposé par Schapire et al. (1997).

AdaBoost

AdaBoost, dont le nom vient de *Adaptive Boosting*, est un algorithme qui permet de construire un classifieur de manière itérative, en forçant un classifieur faible à se concentrer sur les erreurs du modèle grâce à un système de pondération des exemples d'entraînement.

Définition 9.4 (AdaBoost) Supposons un jeu de données de classification binaire $\mathcal{D} = \{(\vec{x}^{\,i}, y^i)\}_{i=1,\dots,n}$, un nombre d'itérations M et un algorithme d'apprentissage. Étant donné un jeu de données $\mathcal{S}$, on note $f_{\mathcal{S}}$ la fonction de décision retournée par cet algorithme. Nous supposons $\mathcal{Y} = \{-1, 1\}$ et que $f_{\mathcal{S}}$ est à valeur dans $\mathcal{Y}$.

Nous appelons *jeu de données pondérées* $\mathcal{D}' = \{(w_i, \vec{x}^{\,i}, y^i)\}_{i=1,\dots,n} \in \mathbb{R}^n \times \mathcal{X}^n \times \mathcal{Y}^n$ un jeu de données $\{(\vec{x}^{\,i}, y^i)\}_{i=1,\dots,n}$ dans lequel un poids w_i a été affecté à la i-ème observation. Nous supposons ici que l'algorithme d'apprentissage que nous utilisons est capable d'intégrer ces pondérations. Dans le cas des arbres de décision, la pondération des exemples d'apprentissage se reflète par leur pondération dans le critère d'impureté ; la décision est également prise par vote majoritaire pondéré.

On appelle *AdaBoost* la procédure de construction d'un ensemble d'apprenants suivante :

1. Initialiser les pondérations $w_1^1, w_2^1, \dots, w_n^1$ à $\frac{1}{n}$.
2. Pour $m = 1, 2, \dots, M$:
   (a) Apprendre sur le jeu de données pondéré $\mathcal{D}_m = \{(w_i^m, \vec{x}^{\,i}, y^i)\}_{i=1,\dots,n}$ la fonction de décision $f_m = f_{\mathcal{D}_m}$
   (b) Calculer l'erreur pondérée de ce modèle :
   $$\epsilon_m = \sum_{i=1}^{m} w_i^m \delta(f_m(\vec{x}^{\,i}), y^i) \tag{9.6}$$
   (c) En déduire la confiance que l'on peut lui associer :
   $$\alpha_m = \frac{1}{2} \log \frac{1 - \epsilon_m}{\epsilon_m}. \tag{9.7}$$
   α_m est d'autant plus élevé que l'erreur globale du modèle est faible : on pourra alors lui faire plus confiance.
   (d) Actualiser les poids, de sorte à donner plus d'importance à un exemple d'entraînement sur lequel f_m se trompe :
   $$w_i^{m+1} = \frac{1}{Z_m} w_i^m e^{-\alpha_m y^i f_m(\vec{x}^{\,i})} \text{ où } Z_m = \sum_{l=1}^{n} w_l^m e^{-\alpha_m y^l f_m(\vec{x}^{\,l})}. \tag{9.8}$$
   Le rôle de Z_m est d'assurer que la somme des coefficients w_i^{m+1} vaut 1.
3. Retourner la fonction de décision finale
   $$f : \vec{x} \mapsto \sum_{m=1}^{M} \alpha_m f_m(\vec{x}).$$

■

Il est classique de considérer comme apprenants faibles pour AdaBoost un type bien particulier d'arbres de décision : les arbres de décision de profondeur 1, appelés *stumps* (qui signifie *souche* en anglais).

Boosting du gradient

AdaBoost se trouve en fait être un cas particulier d'une famille de techniques appelées *boosting du gradient* (*gradient boosting*, ou *GBOOST*), dont le cadre théorique a été développé en 1999 par, d'une part Jerome H. Friedman, et d'autre part Llew Mason, Jonathan Baxter, Peter Bartlett et Marcus Frean (Friedman, 2001; Mason et al., 1999).

Ce cadre permet tout d'abord de mieux comprendre le fonctionnement d'AdaBoost, en considérant la minimisation du risque empirique sur $\mathcal{D}$ et la fonction d'*erreur exponentielle* comme fonction de coût.

Définition 9.5 (Erreur exponentielle) Étant donné un problème de classification, on appelle fonction d'*erreur exponentielle*, ou *exponential loss*, la fonction de coût suivante :

$$\begin{aligned}\{-1,1\} \times \mathbb{R} &\to \mathbb{R} \\ y, f(\vec{x}) &\mapsto e^{-yf(\vec{x})}.\end{aligned}$$

■

Reprenons l'algorithme AdaBoost, et appelons F_m la fonction de décision cumulative $F_m : \vec{x} \mapsto \sum_{k=1}^{m} \alpha_k f_k(\vec{x})$.

À l'étape m, l'erreur exponentielle de F_m sur le jeu d'entraînement vaut

$$\begin{aligned}E_m &= \frac{1}{n}\sum_{i=1}^{n} \exp\left(-y^i \sum_{k=1}^{m-1} \alpha_k f_k(\vec{x}^{\,i})\right) \exp\left(-\alpha_m\, y^i f_m(\vec{x}^{\,i})\right) \\ &= \frac{1}{n}\sum_{i=1}^{n} \exp\left(-y^i F_{m-1}(\vec{x}^{\,i})\right) \exp\left(-\alpha_m\, y^i f_m(\vec{x}^{\,i})\right).\end{aligned}$$

Définissons maintenant $w_i^m = \exp\left(-y^i F_{m-1}(\vec{x}^{\,i})\right)$; alors

$$E_m = \frac{1}{n}\sum_{i=1}^{n} w_i^m \exp\left(-\alpha_m\, y^i f_m(\vec{x}^{\,i})\right).$$

Comme f_m est à valeur dans $\{-1,1\}$, le produit $y^i f_m(\vec{x}^{\,i})$ vaut 1 si la prédiction est correcte et -1 sinon, et donc

$$E_m = \frac{1}{n}\sum_{i=1}^{n} w_i^m e^{-\alpha_m} + \frac{1}{n}\sum_{i:f_m(\vec{x}^{\,i})\neq y^i} w_i^m \left(e^{\alpha_m} - e^{-\alpha_m}\right).$$

Cette erreur est minimale quand α_m a la forme donnée par l'équation 9.7. Ainsi, AdaBoost combine les apprenants faibles de sorte à minimiser, à chaque étape, l'erreur exponentielle du classifieur global.

L'erreur exponentielle peut être remplacée par une autre fonction de coût, telle que l'entropie croisée ou l'erreur quadratique : c'est ce que l'on appelle le *boosting du gradient*. GBOOST est aujourd'hui un des algorithmes les plus populaires en machine learning.

Points clefs

— Les arbres de décision sont des modèles interprétables, qui manient naturellement des variables de plusieurs natures (réelles, discrètes et binaires) et se prêtent aisément à l'apprentissage multi-classe de distributions multi-modales.

— Les arbres de décision ont le grand inconvénient d'être des apprenants faibles, et d'avoir en général une capacité de modélisation trop limitée pour avoir de bonnes performances en pratique.

— Les méthodes ensemblistes permettent de remédier aux limitations des apprenants faibles tels que les arbres de décision, en combinant ces modèles de sorte à compenser leurs erreurs respectives.

— Les méthodes ensemblistes parallèles, telles que le bagging ou les forêts aléatoires, construisent un grand nombre de modèles faibles entraînés sur un échantillonage bootstrap des données.

— Les forêts aléatoires entraînent leurs arbres de façon à ce qu'ils soient indépendants les uns des autres conditionnelement aux données, en sélectionnant aléatoirement les variables à considérer à la création de chaque nœud.

— Les algorithmes de boosting combinent des modèles faibles entraînés séquentiellement pour donner plus d'importance aux exemples d'entraînement sur lesquels les prédictions sont les moins bonnes.

Pour aller plus loin

- Les forêts aléatoires permettent aussi d'établir une *mesure d'importance* de chaque variable, qui se base sur la fréquence à laquelle elle est utilisée comme variable séparatrice parmi tous les arbres de la forêt. Plus précisément, l'importance par permutation mesure la différence de performance entre un arbre évalué sur un jeu de test et un arbre évalué sur ce même jeu de test dans lequel on a permuté aléatoirement les entrées de la j-ème variable : si cette variable est importante, alors le deuxième arbre devrait être moins performant. L'autre mesure parfois utilisée, celle de la diminution du critère d'impureté, fait la somme pondérée de la diminution du critère de Gini pour toutes les coupures de l'arbre réalisées selon la variable dont on mesure l'importance.
- L'implémentation la plus connue de GBOOST, au point d'ailleurs d'en devenir synonyme, est l'implémentation XGBOOST (Chen et Guestrin, 2016), disponible à l'URL `https://github.com/dmlc/xgboost`.

Bibliographie

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 26 :123–140.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1) :5–32.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth International Group, Belmont, CA.

Chen, T. et Guestrin, C. (2016). XGBoost : A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, New York, NY, USA. ACM.

Friedman, J. H. (2001). Greedy function approximation : a gradient boosting machine. *The Annals of Statistics*, 29(5) :1189–1232.

Mason, L., Baxter, J., Bartlett, P., et Frean, M. (1999). Boosting algorithms as gradient descent. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, pages 512–518, Cambridge, MA, USA. MIT Press.

Schapire, R., Freund, Y., Bartlett, P., et Lee, W. S. (1997). Boosting the margin : a new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26 :322–330.

Chapitre 10

Machines à vecteurs de support et méthodes à noyaux

Les *machines à vecteurs de support* (aussi appelées *machines à vecteurs supports*), ou SVM de l'anglais *support vector machines*, sont de puissants algorithmes d'apprentissage automatique. Elles se basent sur un algorithme linéaire proposé par Vladimir Vapnik et Aleksandr Lerner en 1963 (Vapnik et Lerner, 1963), mais permettent d'apprendre bien plus que des modèles linéaires. En effet, au début des années 1990, Vladimir Vapnik, Bernhard Boser, Isabelle Guyon et Corinna Cortes (Boser et al., 1992; Cortes et Vapnik, 1995) ont trouvé comment les étendre efficacement à l'apprentissage de modèles non linéaires grâce à l'*astuce du noyau*. Ce chapitre présente cette approche dans ses différentes versions pour un problème de classification, et introduit ainsi la famille des *méthodes à noyaux*

Objectifs

— Définir un classifieur à large marge dans le cas séparable ;

— Écrire les problèmes d'optimisation primal et dual correspondants ;

— Réécrire ces problèmes dans le cas non séparable ;

— Utiliser l'astuce du noyau pour appliquer une SVM à marge souple dans le cas non linéaire ;

— Définir des noyaux pour des données à valeurs réelles ou des chaînes de caractères.

10.1 Le cas linéairement séparable : SVM à marge rigide

Nous nous plaçons dans ce chapitre dans le cas d'un problème de classification binaire. Dans cette section, nous allons supposer qu'il est possible de trouver un modèle linéaire qui ne fasse pas d'erreur sur nos données : c'est ce qu'on appelle un scénario *linéairement séparable*.

Définition 10.1 (Séparabilité linéaire) Soit $\mathcal{D} = \{(\vec{x}^{\,i}, y^i)\}_{i=1,\dots,n}$ un jeu de données de n observations. Nous supposons que $\vec{x}^{\,i} \in \mathbb{R}^p$ et $y^i \in \{-1, 1\}$. On dit que $\mathcal{D}$ est *linéairement séparable* s'il existe au moins un hyperplan dans $\mathbb{R}^p$ tel que tous les points positifs (étiquetés $+1$) soient d'un côté de cet hyperplan et tous les points négatifs (étiquetés -1) de l'autre. ■

Dans ce cas, il existe en fait *une infinité* d'hyperplans séparateurs qui ne font aucune erreur de classification (voir figure 10.1). Ces hyperplans sont des modèles équivalents du point de vue de la minimisation du risque empirique.

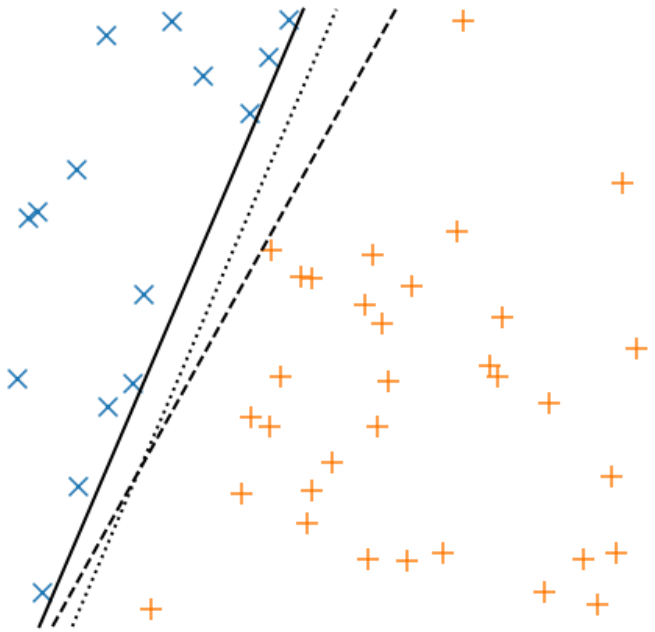


FIGURE 10.1 – Une infinité d'hyperplans (en deux dimensions, des droites) séparent les points négatifs (x) des points positifs (+).

10.1.1 Marge d'un hyperplan séparateur

En l'absence d'information supplémentaire, l'hyperplan séparateur en pointillés sur la figure 10.1 semble préférable. En effet, celui-ci, étant équidistant de l'observation positive la plus proche et de l'observation négative la plus proche, coupe en quelque sorte la poire en deux pour la région située « entre » les points positifs et les points négatifs. La *marge* d'un hyperplan séparateur permet de formaliser cette intuition.

Définition 10.2 (Marge) La *marge* γ d'un hyperplan séparateur est la distance de cet hyperplan à l'observation du jeu d'entraînement la plus proche. ■

L'hyperplan séparateur que nous cherchons est donc celui qui *maximise la marge*. Il y a alors au moins une observation négative et une observation positive qui sont à une distance γ de l'hyperplan séparateur : dans le cas contraire, si par exemple toutes les observations négatives étaient à une distance supérieure à γ de l'hyperplan séparateur, on pourrait rapprocher cet hyperplan des observations négatives et augmenter la marge.

Nous pouvons alors définir, en plus de l'hyperplan séparateur H, les hyperplans H_+ et H_- qui lui sont parallèles et situés à une distance γ de part et d'autre. H_+ contient au moins une observation positive, tandis que H_- contient au moins une observation négative.

Définition 10.3 (Vecteurs de support) On appelle *vecteurs de support* les observations du jeu d'entraînement situés à une distance γ de l'hyperplan séparateur. Elles « soutiennent » les hyperplans H_+ et H_-.

■

C'est de là que vient le nom de la méthode, appelée *Support Vector Machine* ou *SVM* en anglais, et *machine à vecteurs de support* en français. On rencontre aussi parfois le nom de « séparatrice à vaste marge », qui respecte les initiales SVM.

Si l'on venait à déplacer légèrement une observation qui est vecteur de support, cela déplacerait la zone d'indécision et l'hyperplan séparateur changerait. À l'inverse, si l'on déplace légèrement une observation qui n'est pas vecteur de support, H n'est pas affecté : les vecteurs de support sont les observations qui soutiennent la solution.

Toutes les observations positives sont situées à l'extérieur de H_+, tandis que toutes les observations négatives sont situées à l'extérieur de H_-.

Définition 10.4 (Zone d'indécision) On appelle *zone d'indécision* la zone située entre H_- et H_+. Cette zone ne contient aucune observation. ■

La figure 10.2 illustre ces concepts.

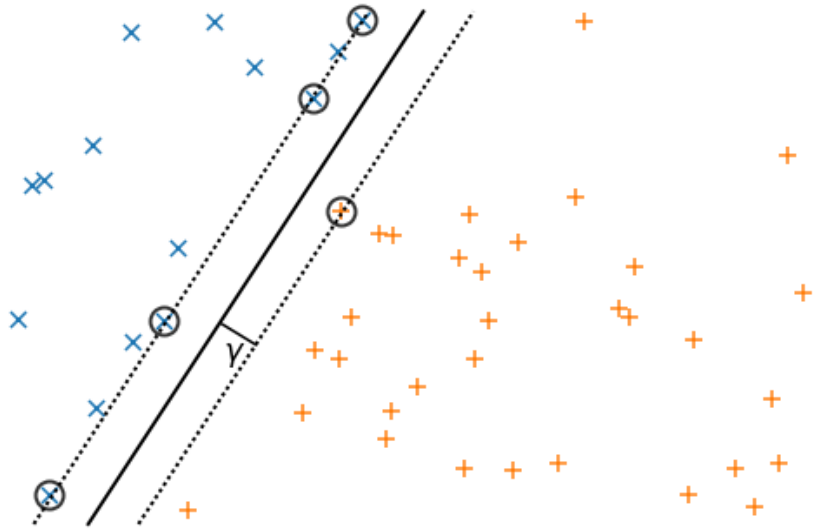


FIGURE 10.2 – La marge γ d'un hyperplan séparateur (ici en trait plein) est sa distance à l'observation la plus proche. Quand cette marge est maximale, au moins une observation négative et une observation positive sont à une distance γ de l'hyperplan séparateur. Les hyperplans (ici en pointillés) parallèles à l'hyperplan séparateur et passant par ces observations définissent la zone d'indécision. Les observations situées sur ces hyperplans (cerclées) sont les vecteurs de support.

10.1.2 Formulation de la SVM à marge rigide

L'équation de l'hyperplan séparateur H que nous cherchons est de la forme $\langle \vec{w}, \vec{x} \rangle + b = 0$, où $\langle , \rangle$ représente le produit scalaire sur $\mathbb{R}^P$. L'hyperplan H_+ lui est parallèle, et a donc pour équation $\langle \vec{w}, \vec{x} \rangle + b =$ constante. Nous pouvons fixer cette constante à 1 sans perdre en généralité : en effet, si nous multiplions $\vec{w}$ et b par une constante, cela n'affectera pas l'équation de H. H_- étant symétrique de H par rapport à H_+, son équation est alors $\langle \vec{w}, \vec{x} \rangle + b = -1$. Enfin, la marge qui est la distance de H à H_+ vaut $\gamma = \frac{1}{||\vec{w}||_2}$.

Les observations positives vérifient donc toutes $\langle \vec{w}, \vec{x} \rangle + b \geq 1$. De même, les points négatifs vérifient tous $\langle \vec{w}, \vec{x} \rangle + b \leq 1$. Ainsi, pour chacun des points de notre jeu d'entraînement, $y^i \left(\langle \vec{w}, \vec{x}^{\,i} \rangle + b \right) \geq 1$. L'égalité est vérifiée pour les vecteurs de support.

Nous cherchons donc ici à maximiser $\frac{1}{||\vec{w}||_2}$ sous les n contraintes $y^i \left(\langle \vec{w}, \vec{x}^{\,i} \rangle + b \right) \geq 1$. Notre problème peut donc se formaliser comme suit :

Définition 10.5 (Formulation primale de la SVM à marge rigide) On appelle *SVM à marge rigide* le problème d'optimisation suivant :

$$\underset{\vec{w} \in \mathbb{R}^p, b \in \mathbb{R}}{\arg\min} \frac{1}{2} ||\vec{w}||_2^2 \text{ t.q. } y^i \left(\langle \vec{w}, \vec{x}^{\,i} \rangle + b \right) \geq 1, i = 1, \ldots, n. \tag{10.1}$$

■

Supposons $\vec{w}^{\,*}, b^*$ solutions de l'équation 10.1. La fonction de décision est alors donnée par

$$f(\vec{x}) = \langle \vec{w}^{\,*}, \vec{x} \rangle + b^*. \tag{10.2}$$

10.1.3 Formulation duale

Le problème défini par l'équation 10.1 est un problème d'optimisation convexe sous n contraintes, chacune des contraintes correspondant à un des points du jeu d'entraînement. De plus, ces contraintes sont toutes affines. Les conditions de Slater nous garantissent donc que la fonction objective de la SVM à marge rigide (minimisée dans l'équation 10.1) est minimisée aux mêmes points que son dual (voir la section A.4.3). Nous pouvons donc en déduire une deuxième formulation équivalente du problème :

Théorème 10.1 (Formulation duale de la SVM à marge rigide) *Le problème défini par l'équation 10.1 est équivalent à :*

$$\begin{aligned} &\max_{\vec{\alpha} \in \mathbb{R}^n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{l=1}^{n} \alpha_i \alpha_l y^i y^l \langle \vec{x}^{\,i}, \vec{x}^{\,l} \rangle \\ &\textit{t. q. } \sum_{i=1}^{n} \alpha_i y^i = 0; \ \alpha_i \geq 0, i = 1, \ldots, n. \end{aligned} \tag{10.3}$$

■

Démonstration. Il s'agit de la formulation duale d'un problème d'optimisation convexe sous contraintes (voir section A.4). Introduisons n multiplicateurs de Lagrange $\{\alpha_i\}_{i=1,\ldots,n}$, un pour chaque contrainte. Le *lagrangien* est donc la fonction

$$\begin{aligned} L : \mathbb{R}^p \times \mathbb{R} \times \mathbb{R}_+^n &\rightarrow \mathbb{R} \\ \vec{w}, b, \vec{\alpha} &\mapsto \frac{1}{2} ||\vec{w}||_2^2 - \sum_{i=1}^{n} \alpha_i \left(y^i \left(\langle \vec{w}, \vec{x}^{\,i} \rangle + b \right) - 1 \right). \end{aligned} \tag{10.4}$$

La *fonction duale de Lagrange* est donc la fonction

$$\begin{aligned} q : \mathbb{R}_+^n &\rightarrow \mathbb{R} \\ \vec{\alpha} &\mapsto \inf_{\vec{w} \in \mathbb{R}^p, b \in \mathbb{R}} L(\vec{w}, b, \vec{\alpha}) \end{aligned} \tag{10.5}$$

Enfin, le problème dual de celui présenté par l'équation 10.1 est donc

$$\max_{\vec{\alpha}\in\mathbb{R}_+^n} \inf_{\vec{w}\in\mathbb{R}^p, b\in\mathbb{R}} \frac{1}{2}\,||\vec{w}||_2^2 - \sum_{i=1}^{n} \alpha_i \left(y^i \left(\langle \vec{w}, \vec{x}^{\,i}\rangle + b\right) - 1\right) \tag{10.6}$$

Le lagrangien est convexe en $\vec{w}$ et est donc minimal quand son gradient en $\vec{w}$ est nul, à savoir quand

$$\vec{w} = \sum_{i=1}^{n} \alpha_i y^i \vec{x}^{\,i}. \tag{10.7}$$

De plus, il est affine en b. Son infimum est donc $-\infty$, sauf si son gradient en b est nul (auquel cas la fonction affine est « plate »), à savoir si

$$\sum_{i=1}^{n} \alpha_i y^i = 0. \tag{10.8}$$

La fonction duale q est donc maximisée dans ce deuxième cas.

En remplaçant $\vec{w}$ par son expression (équation 10.7) dans l'écriture de la fonction duale, l'équation 10.6 peut donc être reformulée comme

$$\max_{\vec{\alpha}\in\mathbb{R}^n} \frac{1}{2}\sum_{i=1}^{n}\sum_{l=1}^{n} \alpha_i\alpha_l y^i y^l \langle \vec{x}^{\,i}, \vec{x}^{\,l}\rangle - \sum_{i=1}^{n} \alpha_i y^i \sum_{l=1}^{n} \alpha_l y^l \langle \vec{x}^{\,l}, \vec{x}^{\,i}\rangle - \sum_{i=1}^{n} \alpha_i y^i b + \sum_{i=1}^{n} \alpha_i$$
$$\text{t. q. } \sum_{i=1}^{n} \alpha_i y^i = 0;\ \ \alpha_i \geq 0, i = 1, \ldots, n.$$

On obtient le résultat recherché en utilisant l'équation 10.8. □

Pour résoudre le problème primal (équation 10.1), on peut donc commencer par résoudre le problème dual (équation 10.3). Supposons $\vec{\alpha}^{\,*}$ solution du problème 10.3. L'équation 10.7 nous donne la solution en $\vec{w}$ de l'équation 10.1 : $\vec{w}^{\,*} = \sum_{i=1}^{n} \alpha_i^* y^i \vec{x}^{\,i}$.

Pour trouver b^*, on peut revenir à la formulation initiale de la SVM : les vecteurs de support positifs sont situés sur l'hyperplan H_+ et vérifient $\langle \vec{w}^{\,*}, \vec{x}^{\,i}\rangle + b^* = 1$.

De plus, étant les observations positives les plus proches de l'hyperplan séparateur H, les vecteurs de support positifs minimisent $\langle \vec{w}^{\,*}, \vec{x}^{\,i}\rangle$. Ainsi

$$b^* = 1 - \min_{i:y^i=+1} \langle \vec{w}^{\,*}, \vec{x}^{\,i}\rangle.$$

Enfin, la *fonction de décision* est donnée par :

$$f(\vec{x}) = \sum_{i=1}^{n} \alpha_i^* y^i \langle \vec{x}^{\,i}, \vec{x}\rangle + b^*. \tag{10.9}$$

Encart

Complexité algorithmique

La formulation primale de la SVM est un problème d'optimisation en $p+1$ dimensions, tandis que la formulation duale est un problème d'optimisation en n dimensions. Si l'on a peu de données et beaucoup de variables, on préférera la formulation duale ; dans le cas inverse, on préférera résoudre le problème primal.

10.1.4 Interprétation géométrique

Les *conditions de Karush-Kuhn-Tucker* (voir section A.4.4) nous permettent de caractériser plus précisément la relation entre $\vec{\alpha}^*$ et $(\vec{w}^*, b^*)$. Appelons ϕ la fonction qui à $\vec{w}$ associe $\frac{1}{2}||\vec{w}||_2^2$, et g_i la fonction qui à $\vec{w}, b$ associe $g_i(\vec{w}, b) = y^i\left(\langle \vec{w}, \vec{x}^{\,i}\rangle + b\right) - 1$. Pour tout $1 \leq i \leq n$, la condition d'*écart complémentaire* signifie que $\alpha_i^* g_i(\vec{w}^*, b^*) = 0$.

Plus précisément, deux cas sont possibles pour chacune des conditions i :

- $\alpha_i^* = 0$: le minimiseur de ϕ vérifie la contrainte et $g_i(\vec{w}^*, b^*) > 0$, autrement dit le point $\vec{x}^{\,i}$ est *à l'extérieur* des hyperplans H_+ ou H_- ;
- $\alpha_i^* > 0$: la contrainte est vérifiée en bordure de la zone de faisabilité, autrement dit à l'égalité $g_i(\vec{w}^*, b^*) = 0$, et $\vec{x}^{\,i}$ est un vecteur de support.

Ainsi, *les vecteurs de support sont les observations du jeu de données correspondant à un multiplicateur de Lagrange α_i^* non nul.*

10.2 Le cas linéairement non séparable : SVM à marge souple

Nos données ne sont généralement pas linéairement séparables. Dans ce cas, quel que soit l'hyperplan séparateur que l'on choisisse, certains des points seront mal classifiés ; d'autres seront correctement classifiés, mais à l'intérieur de la zone d'indécision. Ces concepts sont illustrés sur la figure 10.3.

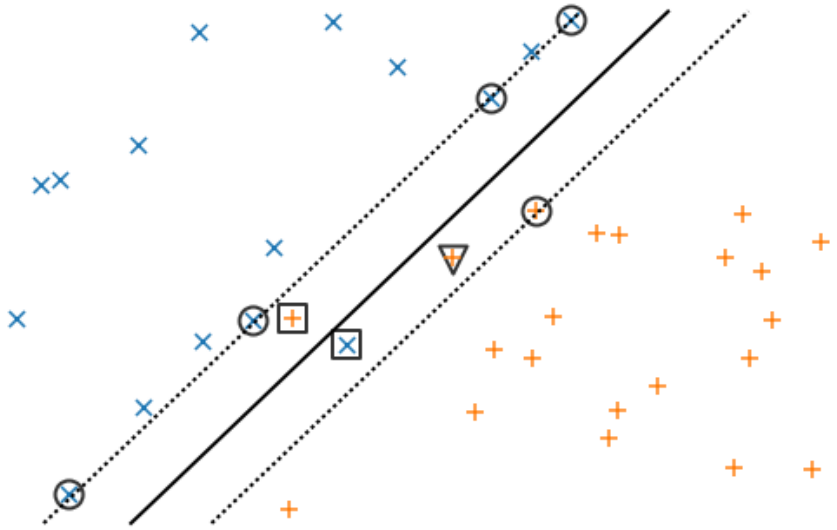


FIGURE 10.3 – Aucun classifieur linéaire ne peut séparer parfaitement ces données. Les observations marquées d'un carré sont des erreurs de classification. L'observation marquée d'un triangle est correctement classifiée mais est située à l'intérieur de la zone d'indécision. Si elle était à sa frontière, autrement dit, si elle était vecteur de support, la marge serait beaucoup plus étroite.

10.2.1 Formulation de la SVM à marge souple

Notre but est maintenant de trouver un compromis entre les erreurs de classification et la taille de la marge. Comme dans la SVM à marge rigide (équation 10.1), nous cherchons à minimiser l'inverse du carré

de la marge $\frac{1}{2}\,||\vec{w}||_2^2$, auquel nous rajoutons un terme d'erreur $C \times \sum_{i=1}^{n} l(f(\vec{x}^{\,i}), y^i)$. Ici $C \in \mathbb{R}^+$ est un hyperparamètre de la SVM, et L représente une fonction de coût :

$$\underset{\vec{w}\in\mathbb{R}^p, b\in\mathbb{R}}{\arg\min}\ \frac{1}{2}\,||\vec{w}||_2^2 + C\sum_{i=1}^{n} L(\langle \vec{w}, \vec{x}^{\,i}\rangle + b, y^i). \tag{10.10}$$

L'hyperparamètre de coût C permet de contrôler l'importance relative de la marge et des erreurs du modèle sur le jeu d'entraînement. Il confère ainsi une certaine *souplesse* à la marge ; on parle alors de *SVM à marge souple.* Nous souhaitons, autant que possible, que toute observation $\vec{x}$ d'étiquette y soit située à l'extérieur de la zone d'indécision, autrement dit vérifie $yf(\vec{x}) \geq 1$. Nous allons donc utiliser l'*erreur hinge* (cf. 2.12) comme fonction de coût. Ainsi l'équation 10.10 peut se réécrire comme suit :

Définition 10.6 (SVM à marge souple) On appelle *SVM à marge souple* la solution du problème d'optimisation suivant :

$$\underset{\vec{w}\in\mathbb{R}^p, b\in\mathbb{R}}{\arg\min}\ \frac{1}{2}\,||\vec{w}||_2^2 + C\sum_{i=1}^{n} \left[1 - y^i f(\vec{x}^{\,i})\right]_+. \tag{10.11}$$

■

Remarque

Cette formulation est celle d'une régularisation de la minimisation d'un risque empirique par un terme de norme ℓ_2. Elle est similaire à la régression ridge (voir le chapitre 6), C étant inversement proportionnel à λ, mais la fonction d'erreur est différente.

Définition 10.7 (Formulation primale de la SVM à marge souple) En introduisant une *variable d'ajustement* (ou *variable d'écart* ; on parlera de *slack variable* en anglais) $\xi_i = \left[1 - y^i f(\vec{x}^{\,i})\right]_+$ pour chaque observation du jeu d'entraînement, le problème d'optimisation 10.11 est équivalent à

$$\begin{aligned} \underset{\vec{w}\in\mathbb{R}^p, b\in\mathbb{R}, \vec{\xi}\in\mathbb{R}^n}{\arg\min}\ & \frac{1}{2}\,||\vec{w}||_2^2 + C\sum_{i=1}^{n} \xi_i \\ \text{t. q. } & y^i\left(\langle \vec{w}, \vec{x}^{\,i}\rangle + b\right) \geq 1 - \xi_i, i = 1, \ldots, n \\ & \xi_i \geq 0, i = 1, \ldots, n. \end{aligned} \tag{10.12}$$

■

10.2.2 Formulation duale

Comme dans le cas de la SVM à marge rigide, il s'agit d'un problème d'optimisation convexe, cette fois sous $2n$ contraintes, toutes affines et les conditions de Slater s'appliquent.

Théorème 10.2 (Formulation duale de la SVM à marge souple) *Le problème 10.12 est équivalent à*

$$\begin{aligned} \max_{\vec{\alpha}\in\mathbb{R}} & \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{l=1}^{n} \alpha_i\alpha_l y^i y^l \langle \vec{x}^{\,i}, \vec{x}^{\,l}\rangle \\ \textit{t. q. } & \sum_{i=1}^{n} \alpha_i y^i = 0;\ 0 \leq \alpha_i \leq C, i = 1, \ldots, n. \end{aligned} \tag{10.13}$$

■

Démonstration. Introduisons $2n$ multiplicateurs de Lagrange $\{\alpha_i, \beta_i\}_{i=1,\dots,n}$ et écrivons le *lagrangien* :

$$\begin{aligned} L : \mathbb{R}^p \times \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^n_+ \times \mathbb{R}^n_+ &\to \mathbb{R} \\ \vec{w}, b, \vec{\xi}, \vec{\alpha}, \vec{\beta} &\mapsto \frac{1}{2}\,||\vec{w}||_2^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i\left(y^i\left(\langle\vec{w}, \vec{x}^{\,i}\rangle + b\right) - 1 + \xi_i\right) \\ &\quad - \sum_{i=1}^{n}\beta_i\xi_i. \end{aligned} \tag{10.14}$$

La *fonction duale de Lagrange* est donc la fonction

$$\begin{aligned} q : \mathbb{R}^n_+ \times \mathbb{R}^n_+ &\to \mathbb{R} \\ \vec{\alpha}, \vec{\beta} &\mapsto \inf_{\vec{w}\in\mathbb{R}^p, b\in\mathbb{R}, \vec{\xi}\in\mathbb{R}^n} L(\vec{w}, b, \vec{\xi}, \vec{\alpha}, \vec{\beta}) \end{aligned} \tag{10.15}$$

Le problème dual de celui présenté par l'équation 10.12 est donc

$$\max_{\vec{\alpha}\in\mathbb{R}^n_+, \vec{\beta}\in\mathbb{R}^n_+} \inf_{\vec{w}\in\mathbb{R}^p, b\in\mathbb{R}, \vec{\xi}\in\mathbb{R}^n} \frac{1}{2}\,||\vec{w}||_2^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i\left(y^i\left(\langle\vec{w}, \vec{x}^{\,i}\rangle + b\right) - 1 + \xi_i\right) - \sum_{i=1}^{n}\beta_i\xi_i. \tag{10.16}$$

Comme dans le cas de la SVM à marge rigide, le lagrangien est minimal quand son gradient en $\vec{w}$ est nul, à savoir quand

$$\vec{w} = \sum_{i=1}^{n}\alpha_i y^i \vec{x}^{\,i}. \tag{10.17}$$

Toujours comme précédemment, il est affine en b et son infimum est donc $-\infty$, sauf si son gradient en b est nul, à savoir si

$$\sum_{i=1}^{n}\alpha_i y^i = 0. \tag{10.18}$$

De plus, il est affine en $\vec{\xi}$ et son infimum est donc $-\infty$, sauf si son gradient en $\vec{\xi}$ est nul, à savoir si

$$\beta_i = C - \alpha_i,\ i = 1, \dots, n. \tag{10.19}$$

La fonction duale q est donc maximisée quand les équations 10.18 et 10.19 sont vérifiés.

En remplaçant $\vec{w}$ par son expression (équation 10.17) dans l'expression de la fonction duale, l'équation 10.16 peut donc être reformulée comme

$$\begin{aligned} &\max_{\vec{\alpha}\in\mathbb{R}^n} -\frac{1}{2}\sum_{i=1}^{n}\sum_{l=1}^{n}\alpha_i\alpha_l y^i y^l \langle\vec{x}^{\,i}, \vec{x}^{\,l}\rangle + \sum_{i=1}^{n}\alpha_i + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}(C - \alpha_i)\xi_i - \sum_{i=1}^{n}\alpha_i\xi_i \\ &\text{t. q. } \sum_{i=1}^{n}\alpha_i y^i = 0;\ \alpha_i \geq 0, i = 1, \dots, n;\ C - \alpha_i \geq 0, i = 1, \dots, n. \end{aligned}$$

□

Remarque

La seule différence avec la formulation duale de la SVM à marge rigide (équation 10.3) est la contrainte $\alpha_i \leq C, i = 1, \dots, n$.

10.2.3 Interprétation géométrique

Comme précédemment, les conditions de Karush-Kuhn-Tucker nous permettent de caractériser plus précisément la relation entre $\vec{\alpha}^*$ et $(\vec{w}^*, b^*)$. Pour chaque observation i, nous avons maintenant deux conditions d'écart complémentaire :

- $\alpha_i^* g_i(\vec{w}^*, b^*) = 0$ et
- $\beta_i^* h_i(\vec{w}^*, b^*) = 0$, ou encore $(C - \alpha_i^*) h_i(\vec{w}^*, b^*) = 0$, où $h_i(\vec{w}, b) = \left[1 - y^i \left(\langle \vec{w}, \vec{x}^{\,i} \rangle + b\right)\right]_+$.

Nous avons ainsi, pour chaque observation i, trois possibilités :

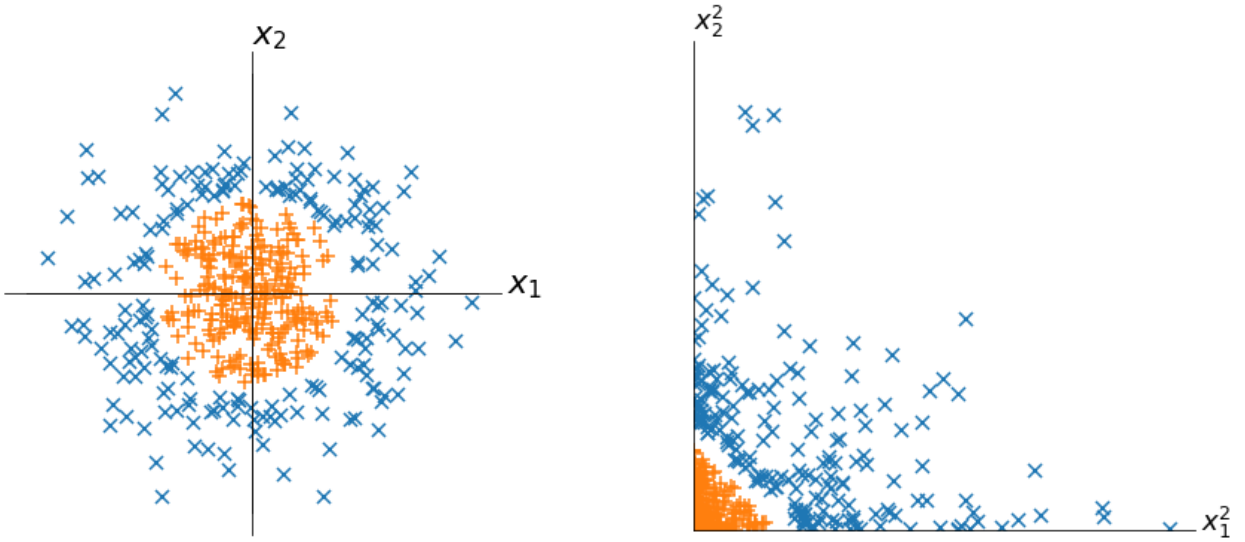
- $\alpha_i^* = 0$: le minimiseur de $\frac{1}{2} ||\vec{w}||_2^2$ vérifie la contrainte et $y^i \left(\langle \vec{w}, \vec{x}^{\,i} \rangle + b\right) > 1$. L'observation $\vec{x}^{\,i}$ est, encore une fois, *à l'extérieur* de la zone d'indécision.
- $0 < \alpha_i^* < C$: comme précédemment, $\vec{x}^{\,i}$ est un vecteur de support situé sur la bordure de la zone d'indécision.
- $\beta_i^* = 0 : \alpha_i^* = C$, auquel cas $\left[1 - y^i \left(\langle \vec{w}, \vec{x}^{\,i} \rangle + b\right)\right]_+ > 0$. Dans ce cas, $\vec{x}^{\,i}$ est *du mauvais côté* de la frontière de la zone d'indécision.

Remarque

Il est possible d'utiliser les SVM pour construire un classifieur multi-classe, grâce à une approche *une-contre-toutes* ou *une-contre-une* (cf. section 2.1.2).

10.3 Le cas non linéaire : SVM à noyau

Il est fréquent qu'une fonction linéaire ne soit pas appropriée pour séparer nos données (voir par exemple la figure 10.4a). Que faire dans ce cas ?



(A) Un cercle semble bien mieux indiqué qu'une droite pour séparer ces données.

(B) Après transformation par l'application $\phi : (x_1, x_2) \mapsto (x_1^2, x_2^2)$, les données sont linéairement séparables dans l'espace de redescription.

FIGURE 10.4 – Transformer les données permet de les séparer linéairement dans un espace de redescription.

10.3.1 Espace de redescription

Dans le cas des données représentées sur la figure 10.4a, un cercle d'équation $x_1^2 + x_2^2 = R^2$ semble bien mieux indiqué qu'une droite pour séparer les deux classes.

Or si la fonction $f : \mathbb{R}^2 \to \mathbb{R}, \vec{x} \mapsto x_1^2 + x_2^2 - R^2$ n'est pas linéaire en $\vec{x} = (x_1, x_2)$, elle est linéaire en (x_1^2, x_2^2). Définissons donc l'application $\phi : \mathbb{R}^2 \to \mathbb{R}^2, (x_1, x_2) \mapsto (x_1^2, x_2^2)$. La fonction de décision f est linéaire en $\phi(\vec{x})$: $f(\vec{x}) = \phi(\vec{x})_1 + \phi(\vec{x})_2 - R^2$. Nous pouvons donc l'apprendre en utilisant une SVM *sur les images des données* par l'application ϕ.

Plus généralement, nous allons maintenant supposer que les observations sont définies sur un espace quelconque $\mathcal{X}$, qui peut être $\mathbb{R}^p$ mais aussi par exemple l'ensemble des chaînes de caractères sur un alphabet donné, l'espace de tous les graphes, ou un espace de fonctions.

Définition 10.8 (Espace de redescription) On appelle *espace de redescription* l'espace de Hilbert $\mathcal{H}$ dans lequel il est souhaitable de redécrire les données, au moyen d'une application $\phi : \mathcal{X} \to \mathcal{H}$, pour y entraîner une SVM sur les images des observations du jeu d'entraînement. ■

La redescription des données dans un espace de Hilbert nous permet d'utiliser un algorithme linéaire, comme la SVM à marge souple, pour résoudre un problème non linéaire.

10.3.2 SVM dans l'espace de redescription

Pour entraîner une SVM sur les images de nos observations dans l'espace de redescription $\mathcal{H}$, il nous faut donc résoudre, d'après l'équation 10.13, le problème suivant :

$$\max_{\vec{\alpha} \in \mathbb{R}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{l=1}^{n} \alpha_i \alpha_l y^i y^l \langle \phi(\vec{x}^{\,i}), \phi(\vec{x}^{\,l}) \rangle_{\mathcal{H}} \tag{10.20}$$
$$\text{t. q. } \sum_{i=1}^{n} \alpha_i y^i = 0; \ 0 \leq \alpha_i \leq C, i = 1, \ldots, n.$$

La fonction de décision sera ensuite donnée par (équation 10.9)

$$f(\vec{x}) = \sum_{i=1}^{n} \alpha_i^* y^i \langle \phi(\vec{x}^{\,i}), \phi(\vec{x}) \rangle_{\mathcal{H}} + b^*. \tag{10.21}$$

10.3.3 Astuce du noyau

Dans les équations 10.20 et 10.21, les images des observations dans $\mathcal{H}$ apparaissent uniquement dans des produits scalaires sur $\mathcal{H}$. Nous pouvons donc, plutôt que la fonction $\phi : \mathcal{X} \to \mathcal{H}$, utiliser la fonction suivante, appelée *noyau* :

$$\begin{array}{rcl} k : \mathcal{X} \times \mathcal{X} & \to & \mathbb{R} \\ \vec{x}, \vec{x}' & \mapsto & \langle \phi(\vec{x}), \phi(\vec{x}') \rangle_{\mathcal{H}}. \end{array}$$

Nous pouvons maintenant définir une *SVM à noyau* :

Définition 10.9 (SVM à noyau) On appelle *SVM à noyau* la solution du problème d'optimisation suivant :

$$\max_{\vec{\alpha} \in \mathbb{R}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{l=1}^{n} \alpha_i \alpha_l y^i y^l k(\vec{x}^{\,i}, \vec{x}^{\,l}) \tag{10.22}$$
$$\text{t. q. } \sum_{i=1}^{n} \alpha_i y^i = 0; \ 0 \leq \alpha_i \leq C, i = 1, \ldots, n.$$

■

La *fonction de décision* sera ensuite donnée par (équation 10.9) :

$$f(\vec{x}) = \sum_{i=1}^{n} \alpha_i^* y^i k(\vec{x}^{\,i}, \vec{x}) + b^*. \tag{10.23}$$

Que ce soit pour entraîner la SVM ou pour l'appliquer, nous n'avons pas besoin de connaître ϕ explicitement, mais il nous suffit de connaître le noyau k. Cela signifie que nous n'avons pas besoin de faire de calcul dans $\mathcal{H}$, qui est généralement de très grande dimension : c'est ce que l'on appelle *l'astuce du noyau.* L'astuce du noyau s'applique de manière générale à d'autres algorithmes d'apprentissage linéaires, comme la régression ridge (cf. section 6.2), l'ACP (cf. section 11.3.1) ou encore la méthode des K-moyennes (cf. section 12.4).

10.3.4 Noyaux

Caractérisation mathématique

Définition 10.10 (Noyau) Nous appelons *noyau* toute fonction k de deux variables s'écrivant sous la forme d'un produit scalaire des images dans un espace de Hilbert de ses variables. Ainsi, un noyau est une fonction continue, symétrique, et semi-définie positive :

$$\forall \quad N \in \mathbb{N}, \quad \forall \quad (\vec{x}^{\,1}, \vec{x}^{\,2}, \dots \vec{x}^{\,N}) \in \mathcal{X}^N \text{ et } (a_1, a_2, \dots a_N) \in \mathbb{R}^N, \sum_{i=1}^{N} \sum_{l=1}^{N} a_i a_l k(\vec{x}^{\,i}, \vec{x}^{\,l}) \geq 0.$$

■

Définition 10.11 (Matrice de Gram) Étant données n observations $\vec{x}^{\,1}, \vec{x}^{\,2}, \dots, \vec{x}^{\,n} \in \mathcal{X}^n$, et un noyau k sur $\mathcal{X}$, on appelle *matrice de Gram* de ces observations la matrice $K \in \mathbb{R}^{n \times n}$ telle que $K_{il} = k(\vec{x}^{\,i}, \vec{x}^{\,l})$. Cette matrice est semi-définie positive. ■

Théorème 10.3 Moore-Aronszajn *Pour toute fonction symétrique semi-définie positive $\kappa : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$, il existe un espace de Hilbert $\mathcal{F}$ et une application $\psi : \mathcal{X} \to \mathcal{F}$ telle que pour tout $\vec{x}, \vec{x}' \in \mathcal{X} \times \mathcal{X}$, $\kappa(\vec{x}, \vec{x}') = \langle \psi(\vec{x}), \psi(\vec{x}') \rangle_{\mathcal{F}}$.* ■

Démonstration. On trouvera une preuve de ce résultat dans l'article original de Nachman Aronszajn, qui l'attribue à E. Hastings Moore (Aronszajn, 1950). □

Intuitivement, un noyau peut être interprété comme un produit scalaire sur un espace de Hilbert, autrement dit comme une fonction qui mesure la *similarité* entre deux objets de $\mathcal{X}$. Ainsi, on peut définir des noyaux en construisant une similarité entre objets, puis en vérifiant qu'elle est semi-définie positive.

Noyaux pour vecteurs réels

Quand $\mathcal{X} = \mathbb{R}^p$, le théorème de Moore-Aronszajn nous permet de définir les noyaux suivants.

Définition 10.12 (Noyau quadratique) On appelle *noyau quadratique* le noyau défini par $k(\vec{x}, \vec{x}') = (\langle \vec{x}, \vec{x}' \rangle + c)^2$ où $c \in \mathbb{R}^+$. ■

L'application correspondant à ce noyau est $\phi : \vec{x} \mapsto (x_1^2, \ldots, x_p^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_1x_p,$ $\ldots \sqrt{2}x_{p-1}x_p, \ldots \sqrt{2c}x_1, \ldots \sqrt{2c}x_p, c)$. ϕ a ainsi valeur dans un espace de dimension $2p + \frac{p(p-1)}{2} + 1$: utiliser k et l'astuce du noyau sera plus efficace que de calculer les images des observations par ϕ avant de leur appliquer une SVM.

Définition 10.13 (Noyau polynomial) On appelle *noyau polynomial* de degré $d \in \mathbb{N}$ le noyau défini par $k(\vec{x}, \vec{x}') = (\langle \vec{x}, \vec{x}' \rangle + c)^d$. ■

Comme pour le noyau quadratique, $c \in \mathbb{R}^+$ est un paramètre permettant d'inclure des termes de degré inférieur à d. Ce noyau correspond à un espace de redescription comptant autant de dimensions qu'il existe de monômes de p variables de degré inférieur ou égal à d, soit $\binom{p+d}{d}$.

Définition 10.14 (Noyau radial gaussien) On appelle *noyau radial gaussien*, ou *noyau RBF* (pour *Radial Basis Function*), de bande passante $\sigma > 0$ le noyau défini par $k(\vec{x}, \vec{x}') = \exp\left(-\frac{||\vec{x}-\vec{x}'||^2}{2\sigma^2}\right)$. ■

Ce noyau correspond à un espace de redescription de dimension *infinie.* En effet, en utilisant le développement en série entière de la fonction exponentielle, on peut le réécrire comme

$$\begin{aligned} k(\vec{x}, \vec{x}') &= \exp\left(-\frac{||\vec{x}||^2}{2\sigma^2}\right) \exp\left(-\frac{\langle \vec{x}, \vec{x}' \rangle}{\sigma^2}\right) \exp\left(-\frac{||\vec{x}'||^2}{2\sigma^2}\right) \\ &= \psi(\vec{x}) \sum_{r=0}^{+\infty} \left(-\frac{\langle \vec{x}, \vec{x}' \rangle^r}{\sigma^{2r} r!}\right) \psi(\vec{x}') = \sum_{r=0}^{+\infty} \left(-\frac{\langle \psi(\vec{x})^{1/r}\vec{x}, \psi(\vec{x}')^{1/r}\vec{x}' \rangle^r}{\sigma^{2r} r!}\right) \end{aligned}$$

avec $\psi : \mathbb{R}^p \to \mathbb{R}, \quad \vec{x} \mapsto \exp\left(-\frac{||\vec{x}||^2}{2\sigma^2}\right)$.

Pour ces trois noyaux, il est plus facile et efficace de calculer directement le noyau entre deux vecteurs de $\mathbb{R}^p$ que de calculer le produit scalaire de leur image dans l'espace de redescription correspondant, qui peut même être de dimension infinie.

Noyaux pour chaînes de caractères

L'astuce du noyau nous permet aussi de travailler sur des données complexes sans avoir à les exprimer tout d'abord en une représentation vectorielle de longueur fixe. C'est le cas en particulier pour les données représentées par des *chaînes de caractères,* comme du texte ou des séquences biologiques telles que de l'ADN (définies sur un alphabet de 4 lettres correspondant aux 4 bases nucléiques) ou des protéines (définies sur un alphabet de 21 acides aminés.)

Étant donné un alphabet $\mathcal{A}$, nous utilisons maintenant $\mathcal{X} = \mathcal{A}^*$ (c'est-à-dire l'ensemble des chaînes de caractères définies sur $\mathcal{A}$.) La plupart des noyaux sur $\mathcal{X}$ sont définis en utilisant l'idée que plus deux chaînes x et x' ont de sous-chaînes en commun, plus elles sont semblables. Étant donnée une longueur $k \in \mathbb{N}$ de sous-chaînes, nous transformons une chaîne x en un vecteur de longueur $|\mathcal{A}|^k$ grâce à l'application $\phi : x \mapsto (\psi_u(x))_{u \in \mathcal{A}^k}$, où $\psi_u(x)$ est le nombre d'occurrences de u dans x. ψ peut être modifiée pour permettre les alignements inexacts, ou autoriser en les pénalisant les « trous » (ou *gaps*.)

On peut alors définir le *noyau pour chaînes de caractères* suivant :

$$\begin{aligned} k : \mathcal{A}^* \times \mathcal{A}^* &\to \mathbb{R} \\ x, x' &\mapsto \sum_{u \in \mathcal{A}^k} \psi_u(x)\psi_u(x'). \end{aligned}$$

Formellement, ce noyau nécessite de calculer une somme sur $|\mathcal{A}^k| = |\mathcal{A}|^k$ termes. Cependant, il peut être calculé de manière bien plus efficace en itérant uniquement sur les $(|x| + 1 - k)$ chaînes de longueur

k présentes dans x, les autres termes de la somme valant nécessairement 0. Il s'agit alors d'un calcul en $\mathcal{O}(|x| + |x'|)$.

Exemple

Dans le cas des protéines humaines, si l'on choisit $k = 8,$ on remplace ainsi un calcul dans un espace de redescription de dimension supérieure à 37 milliards (21^8) par une somme sur moins de 500 termes (la longueur moyenne d'une protéine humaine étant de 485 acides aminés.)

Ces idées peuvent aussi être appliquées à la définition de *noyaux pour graphes*, en remplaçant les sous-chaînes de caractères par des sous-graphes. Cependant, le problème de l'isomorphisme de sous-graphes est NP-complet dans le cas général, ce qui limite le type de sous-graphes que l'on peut considérer.

Noyaux pour ensembles

Certaines des similarités entre ensembles que nous avons définies à la section 8.3.3 sont en fait des noyaux.

Théorème 10.4 *La similarité de Jaccard et la similarité MinMax sont des noyaux.* ■

Démonstration. La preuve de ce théorème repose sur un résultat de Gower (1971). □

10.4 Régression ridge à noyau

Théorème 10.5 *L'astuce du noyau s'applique aussi à la régression ridge linéaire.* ■

Démonstration. Rappelons que la fonction de prédiction de la régression ridge est de la forme

$$f(\vec{x}) = \langle \vec{x}, \vec{\beta}^* \rangle, \tag{10.24}$$

où $\vec{\beta}^*$ est donné par l'équation 6.7 :

$$\vec{\beta}^* = \left(\lambda I_p + X^\top X\right)^{-1} X^\top \vec{y}. \tag{10.25}$$

Ici $X \in \mathbb{R}^{n \times p}$ est la matrice de design potentiellement augmentée d'une colonne de 1 selon la transformation 5.5.

L'expression 10.25 peut se réécrire, en multipliant à gauche par $\left(\lambda I_p + X^\top X\right)$, comme

$$\vec{\beta}^* = X^\top \vec{\alpha} \text{ avec } \vec{\alpha} = \frac{1}{\lambda}\left(y - X\vec{\beta}^*\right).$$

Ainsi $\lambda\vec{\alpha} = y - XX^\top\vec{\alpha}$ et donc

$$\vec{\alpha} = \left(\lambda I_n + XX^\top\right)^{-1} y.$$

L'équation 10.24 peut donc s'écrire

$$f(\vec{x}) = \vec{x}X^\top\vec{\alpha} = xX^\top(\lambda I_n + XX^\top)^{-1}y. \tag{10.26}$$

Soit maintenant $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ un noyau. Il existe un espace de Hilbert $\mathcal{H}$ et une application $\phi : \mathcal{X} \to \mathcal{H}$ telle que, pour tout $\vec{x}, \vec{x}' \in \mathcal{X} \times \mathcal{X}$, $k(\vec{x}, \vec{x}') = \langle \vec{x}, \vec{x}' \rangle_{\mathcal{H}}$.

Appliquer la régression ridge aux images des observations dans $\mathcal{H}$ revient à calculer

$$f_\phi(\vec{x}) = \phi(\vec{x})\Phi^\top(\lambda I_n + \Phi\Phi^\top)^{-1}y,$$

où $\Phi \in \mathbb{R}^{n\times d}$ est la matrice dont la i-ème ligne est l'image par ϕ dans $\mathcal{H}$ (supposé ici de dimension d) de $\vec{x}^{\,i}$. On peut réécrire cette fonction comme

$$f_\phi(\vec{x}) = \kappa(\lambda I_n + K)^{-1}y, \tag{10.27}$$

où $\kappa \in \mathbb{R}^n$ est le vecteur dont la i-ème entrée est

$$\kappa_i = \langle\phi(\vec{x}), \phi(\vec{x}^{\,i})\rangle_\mathcal{H} = k(\vec{x}, \vec{x}^{\,i})$$

et $K \in \mathbb{R}^{n\times n}$ est la matrice dont l'entrée K_{il} est

$$K_{il} = \langle\phi(\vec{x}^{\,i}), \phi(\vec{x}^{\,l})\rangle_\mathcal{H} = k(\vec{x}^{\,i}, \vec{x}^{\,l}).$$

Il n'est donc pas nécessaire d'utiliser ϕ et $\mathcal{H}$ explicitement pour entraîner une régression ridge dans l'espace de redescription $\mathcal{H}$ défini par un noyau. □

Points clefs

- La SVM à marge souple est un algorithme linéaire de classification binaire qui cherche à maximiser la séparation entre les classes, formalisée par le concept de marge, tout en contrôlant le nombre d'erreurs de classification sur le jeu d'entraînement.
- L'astuce du noyau permet d'appliquer efficacement cet algorithme, ainsi que d'autres algorithmes linéaires comme la régression ridge, dans un espace de redescription, sans calculer explicitement l'image des observations dans cet espace.
- Les noyaux quadratique, polynomial et radial gaussien permettent d'utiliser des espaces de redescription de dimensions de plus en plus grandes.

Pour aller plus loin

- Le site web `http://www.kernel-machines.org` propose de nombreuses ressources bibliographiques et logicelles autour des méthodes à noyaux.
- De nombreux ouvrages ont été consacrés aux SVM ; parmi eux, on pourra se référer à Schölkopf et Smola (2002), ainsi qu'au chapitre sur les SVM de Cherkassky et Mulier (1998), à Vert et al. (2004), ou au tutoriel de Burges (1998).
- Les SVM ont été étendues aux problèmes de régression. Il s'agit alors d'un problème de régression linéaire régularisée par une norme ℓ_2, mais avec une perte ϵ-insensible (voir section 2.4.3) comme fonction de coût. Elles sont alors parfois appelées *SVR* pour *Support Vector Regression*. Pour plus de détails, on pourra se référer au tutoriel de Smola et Schölkopf (1998).
- Une alternative aux solveurs d'optimisation quadratique générique est l'algorithme SMO (*Sequential Minimal Optimization*). Proposé par John Platt (1998), il a participé à l'engouement pour les SVM en accélérant leur optimisation. C'est cet algorithme qui est implémentée dans la librairie LibSVM (Chang et Lin, 2008), qui reste une des plus utilisées pour entraîner les SVM ; scikit-learn ou Shogun, par exemple, la réutilisent.

- La fonction de décision des SVM ne modélise pas une probabilité. Platt (1999) propose d'apprendre une fonction sigmoïde qui convertit ces prédictions en probabilité.
- Il est possible d'étendre les SVM afin de modéliser des observations appartenant à une seule et même classe (sans exemples négatifs). Cette méthode permet ensuite de classifier de nouvelles observations selon qu'elles appartiennent ou non à la classe, et ainsi être utilisées pour détecter des anomalies, autrement dit des observations qui n'appartiennent pas à la classe. Il en existe deux variantes : celle appelée *One-class SVM* (Schölkopf et al., 2000), qui sépare les observations de l'origine ; et celle appelée *Support Vector Data Description* (Tax et Duin, 2004), qui permet de trouver une frontière sphérique (dans l'espace de redescription, dans le cas de la version à noyau) qui enveloppe de près les données.
- Pour plus de détails sur les noyaux pour chaînes de caractères et leur utilisation en bioinformatique, on pourra se référer au chapitre de Saunders et Demco (2007). En ce qui concerne les noyaux pour graphe, le chapitre 2 de la thèse de Benoît Gaüzère (2013) donne un bon aperçu du domaine.

Bibliographie

Aronszajn, N. (1950). Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68(3) :337–404.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth annual Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, Pennsylvania, United States. ACM.

Burges, C. J. C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2 :121–167.

Chang, C.-C. et Lin, C.-J. (2008). LibSVM : A library for support vector machines. `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`.

Cherkassky, V. et Mulier, F. (1998). *Learning from Data : Concepts, Theory, and Methods*. Wiley, New York.

Cortes, C. et Vapnik, V. (1995). Support vector networks. *Machine Learning*, 20 :273–297.

Gaüzère, B. (2013). *Application des méthodes à noyaux sur graphes pour la prédiction des propriétés des molécules.* Thèse de doctorat, Université de Caen.

Gower, J. C. (1971). A general coefficient of similarity and some of its properties. *Biometrics*, pages 857–871.

Platt, J. C. (1998). Sequential minimal optimization : a fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft Research.

Platt, J. C. (1999). Probabilities for support vector machines. In *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, Cambridge, MA.

Saunders, C. et Demco, A. (2007). Kernels for strings and graphs. In *Perspectives of Neural-Symbolic Integration*, Studies in Computational Intelligence, pages 7–22. Springer, Berlin, Heidelberg.

Schölkopf, B., Williamson, R. C., Smola, A. J., Shawe-Taylor, J., et Platt, J. C. (2000). Support vector method for novelty detection. In *Advances in neural information processing systems*, pages 582–588.

Schölkopf, B. et Smola, A. J. (2002). *Learning with Kernels : Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA.

Smola, A. J. et Schölkopf, B. (1998). A tutorial on support vector regression. *NeuroCOLT*, TR-1998-030.

Tax, D. M. et Duin, R. P. (2004). Support vector data description. *Machine learning*, 54(1) :45–66.

Vapnik, V. et Lerner, A. (1963). Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24.

Vert, J.-P., Tsuda, K., et Schölkopf, B. (2004). A primer on kernel methods. In *Kernel Methods in Computational Biology*, pages 35–70. MIT Press, Cambridge, MA.

Chapitre 11

Réduction de dimension

Dans certaines applications, le nombre de variables p utilisé pour représenter les données est très élevé. C'est le cas par exemple du traitement d'images haute-résolution, dans lequel chaque pixel peut être représenté par plusieurs variables, ou de l'analyse de données génomiques, où des centaines de milliers de positions du génome peuvent être caractérisées.

Bien qu'une représentation des données contenant plus de variables soit intuitivement plus riches, il est plus difficile d'apprendre un modèle performant dans ces circonstances. Dans ce chapitre, nous en détaillerons les raisons avant d'explorer une série de méthodes, supervisées ou non, pour réduire ce nombre de variables afin d'obtenir des représentations plus compactes et des modèles plus robustes.

Objectifs

— expliquer quel peut être l'intérêt de réduire la dimension d'un jeu de données ;

— faire la différence entre la sélection de variables et l'extraction de variables ;

— connaître les principales techniques pour ces deux approches ;

— comprendre l'analyse en composantes principales à partir d'une définition de maximisation de la variance et comme une technique de factorisation de matrice.

11.1 Motivation

Le but de la réduction de dimension est de transformer une représentation $X \in \mathbb{R}^{n \times p}$ des données en une représentation $X^* \in \mathbb{R}^{n \times m}$ où $m \ll p$. Les raisons de cette démarche sont multiples, et nous les détaillons dans cette section.

11.1.1 Visualiser les données

Bien que le but de l'apprentissage automatique soit justement d'automatiser le traitement des données, il est essentiel pour appliquer les méthodes présentées dans cet ouvrage à bon escient de bien comprendre le problème particulier que l'on cherche à résoudre et les données dont on dispose. Cela permet de mieux définir les variables à utiliser, d'éliminer éventuellement les données aberrantes, et de guider le choix des

algorithmes. Dans ce but, il est très utile de pouvoir *visualiser* les données ; mais ce n'est pas tâche aisée avec p variables. Ainsi, limiter les variables à un faible nombre de dimensions permet de visualiser les données plus facilement, quitte à perdre un peu d'information lors de la transformation.

11.1.2 Réduire les coûts algorithmiques

Cet aspect-là est purement computationnel : réduire la dimension des données permet de réduire d'une part l'espace qu'elles prennent en mémoire et d'autre part les temps de calcul. De plus, si certaines variables sont inutiles, ou redondantes, il n'est pas nécessaire de les obtenir pour de nouvelles observations : cela permet de réduire le coût d'acquisition des données.

11.1.3 Améliorer la qualité des modèles

Notre motivation principale pour réduire la dimension des données est de pouvoir construire de meilleurs modèles d'apprentissage supervisé. Par exemple, un modèle paramétrique construit sur un plus faible nombre de variables est moins susceptible de sur-apprendre, comme nous l'avons vu avec le lasso au chapitre 6.

De plus, si certaines des variables mesurées ne sont pas pertinentes pour le problème d'apprentissage qui nous intéresse, elles peuvent induire l'algorithme d'apprentissage en erreur. Prenons comme exemple l'algorithme des plus proches voisins (chapitre 8), qui base ses prédictions sur les étiquettes des exemples d'entraînement les plus proches de l'observation à étiqueter. Supposons que l'on utilise une distance de Minkowski ; toutes les variables ont alors la même importance dans le calcul des plus proches voisins. Ainsi les variables qui ne sont pas pertinentes peuvent biaiser la définition du plus proche voisin, et introduire du bruit dans le modèle, comme illustré sur la figure 11.1.

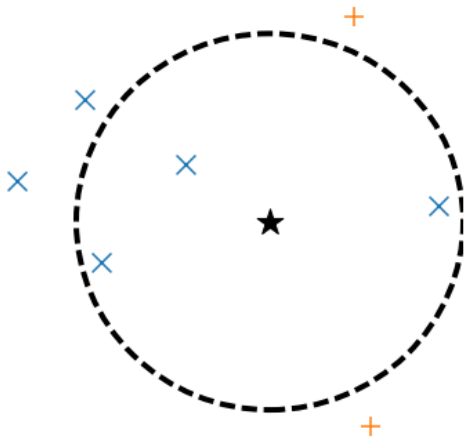


FIGURE 11.1 – En utilisant les deux dimensions, les trois plus proches voisins de l'étoile sont majoritairement des (x). En utilisant seulement la variable en abscisse, ses trois plus proches voisins sont majoritairement des (+). Si la variable en ordonnée n'est pas pertinente, elle fausse le résultat de l'algorithme des trois plus proches voisins.

Enfin, nous faisons face en haute dimension à un phénomène connu sous le nom de *fléau de la dimension*, ou *curse of dimensionality* en anglais. Ce terme qualifie le fait que les intuitions développées en faible dimension ne s'appliquent pas nécessairement en haute dimension.

En effet, en haute dimension, les exemples d'apprentissage ont tendance à tous être éloignés les uns des autres. Pour comprendre cette assertion, plaçons-nous en dimension p et considérons l'hypersphère $\mathcal{S}(\vec{x}, R)$ de rayon $R \in \mathbb{R}_+^*$ centrée sur une observation $\vec{x}$, ainsi que l'hypercube $\mathcal{C}(\vec{x}, R)$ circonscrit à cette hypersphère. Le volume de $\mathcal{S}(\vec{x})$ vaut $\frac{2R^p\pi^{p/2}}{p\Gamma(p/2)}$, tandis que celui de $\mathcal{C}(\vec{x}, R)$, dont le côté a pour longueur $2R$, vaut 2^pR^p. Ainsi

$$\lim_{p\to\infty} \frac{\text{Vol}(\mathcal{C}(\vec{x}, R))}{\text{Vol}(\mathcal{S}(\vec{x}, R))} = 0.$$

Cela signifie que la probabilité qu'un exemple situé dans $\mathcal{C}(\vec{x}, R)$ appartienne à $\mathcal{S}(\vec{x}, R)$, qui vaut $\frac{\pi}{4} \approx 0.79$ lorsque $p = 2$ et $\frac{\pi}{6} \approx 0.52$ lorsque $p = 3$ (cf. figure 11.2), devient très faible quand p est grand : les données ont tendance à être éloignées les unes des autres.

Cela implique que les algorithmes développés en utilisant une notion de similarité, comme les plus proches voisins, les arbres de décision ou les SVM, ne fonctionnent pas nécessairement en grande dimension. Ainsi, réduire la dimension peut être nécessaire à la construction de bons modèles.

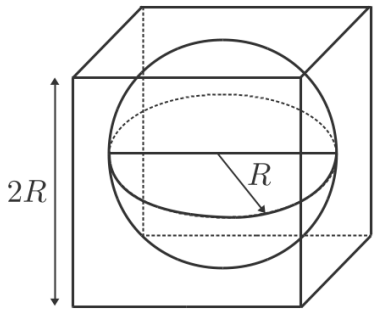


FIGURE 11.2 – Ici en dimension 3, on considère la proportion du volume du cube de côté $a = 2R$ située à l'intérieur de la sphère de rayon R inscrite dans ce cube.

Deux possibilités s'offrent à nous pour réduire la dimension de nos données :
— la *sélection de variables*, qui consiste à éliminer un nombre $p - m$ de variables de nos données ;
— l'*extraction de variables*, qui consiste à *créer* m nouvelles variables à partir des p variables dont nous disposons initialement.

La suite de ce chapitre détaille ces deux approches.

11.2 Sélection de variables

Les approches de *sélection de variables* consistent à choisir m variables à conserver parmi les p existantes, et à ignorer les $p-m$ autres. On peut les séparer en trois catégories : les méthodes de *filtrage*, les méthodes de *conteneur*, et les méthodes *embarquées*. Les méthodes que nous allons voir ici sont des méthodes supervisées : nous supposons disposer d'un jeu de données étiqueté $\mathcal{D} = \{(\vec{x}^{\,i}, y^i)\}_{i=1,\dots,n}$ où $\vec{x}^{\,i} \in \mathbb{R}^p$.

11.2.1 Méthodes de filtrage

La sélection de variable par *filtrage* consiste à appliquer un *critère de sélection* indépendamment à chacune des p variables ; il s'agit de quantifier la pertinence de la p-ème variable du jeu de donnée par rapport à y.

Quelques exemples d'un tel critère sont la corrélation avec l'étiquette, un test statistique comme celui du χ^2 dans le cas d'un problème de classification, ou l'information mutuelle.

La *corrélation* entre la variable j et l'étiquette y se calcule comme celle entre une étiquette prédite et une étiquette réelle (cf. équation 3.2) :

$$R_j = \frac{\sum_{i=1}^{n} \left(y^i - \frac{1}{n}\sum_{i=1}^{n} y^i\right)\left(x_j^i - \frac{1}{n}\sum_{i=1}^{n} f(\vec{x}^{\,i})\right)}{\sqrt{\sum_{i=1}^{n}\left(y^i - \frac{1}{n}\sum_{i=1}^{n} y^i\right)^2}\sqrt{\sum_{i=1}^{n}\left(x_j^i - \frac{1}{n}\sum_{l=1}^{n} x_j^l\right)^2}}. \tag{11.1}$$

Définition 11.1 (Information mutuelle) L'*information mutuelle* entre deux variables aléatoires X_j et Y mesure leur dépendance au sens probabiliste ; elle est nulle si et seulement si les variables sont indépendantes, et croît avec leur degré de dépendance. Elle est définie, dans le cas discret, par

$$I(X_j, Y) = \sum_{x_j, y} \mathbb{P}(X_j = x_j, Y = y) \log \frac{\mathbb{P}(X_j = x_j, Y = y)}{\mathbb{P}(X_j = x_j)\mathbb{P}(Y = y)},$$

et dans le cas continu par

$$I(X_j, Y) = \int_{x_j}\int_{y} p(x_j, y) \log \frac{p(x_j, y)}{p(x_j)p(y)} \mathrm{d}x_j \, \mathrm{d}y.$$

■

L'estimateur de Kozachenko-Leonenko est l'un des plus fréquemment utilisés pour estimer l'information mutuelle (Kozachenko et Leonenko, 1987).

Les méthodes de filtrage souffrent de traiter les variables individuellement : elles ne peuvent pas prendre en compte leurs effets combinés. Un exemple classique pour illustrer ce problème est celui du « ou exclusif » (XOR) : prise individuellement, x_1 (resp. x_2) est décorrélée de $y = x_1$ XOR x_2, alors qu'ensemble, ces deux variables expliqueraient parfaitement l'étiquette y.

11.2.2 Méthodes de conteneur

Les méthodes de *conteneur*, ou *wrapper methods* en anglais, consistent à essayer de déterminer le meilleur sous-ensemble de variables *pour un algorithme d'apprentissage donné.* On parle alors souvent non pas de sélection de variables mais de *sélection de sous-ensemble*, ou *subset selection* en anglais.

Ces méthodes sont le plus souvent des méthodes heuristiques. En effet, il est généralement impossible de déterminer la performance d'un algorithme sur un jeu de données sans l'avoir entraîné. Il n'est pas raisonnable d'adopter une stratégie exhaustive, sauf pour un très faible nombre de variables, car le nombre de sous-ensembles de p variables est de $2^p - 1$ (en excluant $\emptyset$). On adoptera donc une approche gloutonne.

Quelques exemples de ces approches sont la *recherche ascendante*, la *recherche descendante*, et la *recherche flottante* que nous détaillons ci-dessous.

Étant donnés un jeu de données $\mathcal{D} = (X, \vec{y})$ où $X \in \mathbb{R}^{n\times p}$, un sous-ensemble de variables $\mathcal{E} \subset \{1, 2, \ldots, p\}$, et un algorithme d'apprentissage, on notera $X_\mathcal{E} \in \mathbb{R}^{n\times|\mathcal{E}|}$ la matrice X restreinte aux variables apparaissant dans $\mathcal{E}$, et $E_\mathcal{D}(\mathcal{E})$ l'estimation de l'erreur de généralisation de cet algorithme d'apprentissage, entraîné sur $(X_\mathcal{E}, \vec{y})$. Cette estimation est généralement obtenue sur un jeu de test ou par validation croisée.

La recherche ascendante consiste à partir d'un ensemble de variables vide, et à lui ajouter variable par variable celle qui améliore au mieux sa performance, jusqu'à ne plus pouvoir l'améliorer.

Définition 11.2 (Recherche ascendante) On appelle *recherche ascendante*, ou *forward search* en anglais, la procédure gloutonne de sélection de variables suivante :

1. Initialiser $\mathcal{F} = \emptyset$
2. Trouver la meilleure variable à ajouter à $\mathcal{F}$:
$$j^* = \underset{j \in \{1,\dots,p\} \setminus \mathcal{F}}{\arg\min} E_{\mathcal{D}}(\mathcal{F} \cup \{j\})$$
3. Si $E_{\mathcal{D}}(\mathcal{F} \cup \{j\}) > E_{\mathcal{D}}(\mathcal{F})$: s'arrêter
sinon : $\mathcal{F} \leftarrow \mathcal{F} \cup \{j\}$; recommencer 2-3.

■

Dans le pire des cas (celui où on devra itérer jusqu'à ce que $\mathcal{F} = \{1, 2, \dots, p\}$), cet algorithme requiert de l'ordre de $\mathcal{O}(p^2)$ évaluations de l'algorithme d'apprentissage sur un jeu de données, ce qui peut être intensif, mais est bien plus efficace que $\mathcal{O}(2^p)$ comme requis par l'approche exhaustive.

À l'inverse, la recherche descendante consiste à partir de l'ensemble de toutes les variables, et à lui retirer variable par variable celle qui améliore au mieux sa performance, jusqu'à ne plus pouvoir l'améliorer.

Définition 11.3 (Recherche descendante) On appelle *recherche descendante*, ou *backward search* en anglais, la procédure gloutonne de sélection de variables suivante :

1. Initialiser $\mathcal{F} = \{1, \dots, p\}$
2. Trouver la meilleure variable à retirer de $\mathcal{F}$:
$$j^* = \underset{j \in \mathcal{F}}{\arg\min} E_{\mathcal{D}}(\mathcal{F} \setminus \{j\})$$
3. Si $E_{\mathcal{D}}(\mathcal{F} \setminus \{j\}) > E_{\mathcal{D}}(\mathcal{F})$: s'arrêter
sinon : $\mathcal{F} \leftarrow \mathcal{F} \setminus \{j\}$; recommencer 2-3.

■

L'avantage de l'approche descendante sur l'approche ascendante est qu'elle fournit nécessairement un sous-ensemble de variables meilleur que l'intégralité des variables. En effet, ce n'est pas parce qu'on ne peut pas, à une étape de la méthode ascendante, trouver de variable à ajouter à $\mathcal{F}$, que la performance de l'algorithme est meilleure sur $(X_{\mathcal{F}}, \vec{y})$ que sur $(X, \vec{y})$.

La recherche flottante permet d'explorer un peu différemment l'espace des possibles en combinant les deux approches.

Définition 11.4 (Recherche flottante) Étant donné deux paramètres entiers strictement positifs q et r, on appelle *recherche flottante*, ou *floating search* en anglais, la procédure gloutonne de sélection de variables suivante :

1. Initialiser $\mathcal{F} = \emptyset$
2. Trouver les q meilleures variables à ajouter à $\mathcal{F}$:
$$\mathcal{S}^* = \underset{\mathcal{S} \subseteq \{1,\dots,p\} \setminus \mathcal{F},\ |\mathcal{S}| = q}{\arg\min} E_{\mathcal{D}}(\mathcal{F} \cup \mathcal{S})$$
3. Si $E_{\mathcal{D}}(\mathcal{F} \cup \mathcal{S}) < E_{\mathcal{D}}(\mathcal{F})$: $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{S}$
4. Trouver les r meilleures variables à retirer de $\mathcal{F}$:
$$\mathcal{S}^* = \underset{\mathcal{S} \subseteq \{1,\dots,p\} \setminus \mathcal{F},\ |\mathcal{S}| = r}{\arg\min} E_{\mathcal{D}}(\mathcal{F} \setminus \mathcal{S})$$
5. Si $E_{\mathcal{D}}(\mathcal{F} \setminus \mathcal{S}) > E_{\mathcal{D}}(\mathcal{F})$: s'arrêter
sinon : $\mathcal{F} \leftarrow \mathcal{F} \setminus \mathcal{S}$; recommencer 2-5.

■

11.2.3 Méthodes embarquées

Enfin, les *méthodes embarquées*, ou *embedded approaches* en anglais, apprennent en même temps que le modèle les variables à y inclure. Il s'agit généralement de modèles paramétriques parcimonieux, c'est-à-dire tels que les coefficients affectés à certaines variables soient nuls. Ces variables sont alors éliminées du modèle. L'exemple le plus prégnant de ces techniques est le lasso, que nous avons vu dans la section 6.3.

Dans le même ordre d'idée, la mesure d'importance des variables dans les forêts aléatoires (voir chapitre 9) peut être utilisée pour décider quelles variables éliminer.

11.3 Extraction de variables

11.3.1 Analyse en composantes principales

La méthode la plus classique pour réduire la dimension d'un jeu de données par extraction de variables est l'*analyse en composantes principales*, ou *ACP*. On parle aussi souvent de *PCA*, de son nom anglais *Principal Component Analysis*.

Maximisation de la variance

L'idée centrale d'une ACP est de représenter les données de sorte à maximiser leur variance selon les nouvelles dimensions, afin de pouvoir continuer à distinguer les exemples les uns des autres dans leur nouvelle représentation (cf. figure 11.3).

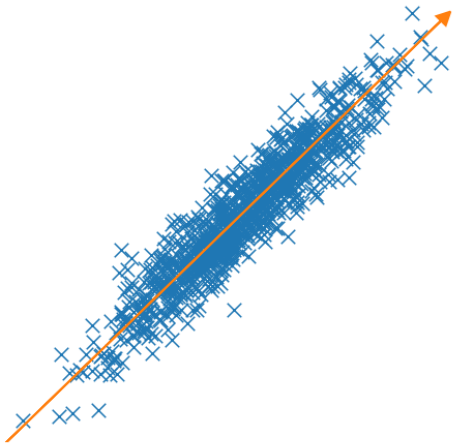


FIGURE 11.3 – La variance des données est maximale selon l'axe indiqué par une flèche.

Formellement, une nouvelle représentation de $\mathcal{X}$ est définie par une base orthonormée sur laquelle projeter la matrice de données X.

Définition 11.5 (Analyse en composantes principales) Une *analyse en composantes principales*, ou *ACP*, de la matrice $X \in \mathbb{R}^{n \times p}$ est une transformation linéaire orthogonale qui permet d'exprimer X dans une nouvelle base orthonormée, de sorte que la plus grande variance de X par projection s'aligne sur le premier axe de cette nouvelle base, la seconde plus grande variance sur le deuxième axe, et ainsi de suite.

Les axes de cette nouvelle base sont appelés les *composantes principales*, abrégées en *PC* pour *Principal Components*. ■

Attention

Dans la suite de cette section, nous supposons que les variables ont été *standardisées* de sorte à toutes avoir une moyenne de 0 et une variance de 1, pour éviter que les variables qui prennent de grandes valeurs aient plus d'importance que celles qui prennent de faibles valeurs. C'est un pré-requis de l'application de l'ACP. Cette standardisation s'effectue en centrant la moyenne et en réduisant la variance de chaque variable :

$$x_j^i \leftarrow \frac{x_j^i - \bar{x_j}}{\sqrt{\frac{1}{n}\sum_{l=1}^{n}(x_j^l - \bar{x_j})^2}}, \tag{11.2}$$

où $\bar{x_j} = \frac{1}{n}\sum_{l=1}^{n} x_j^l$. On dira alors que X est *centrée* : chacune de ses colonnes a pour moyenne 0.

Théorème 11.1 *Soit $X \in \mathbb{R}^{n\times p}$ une matrice centrée de covariance $\Sigma = \frac{1}{n}X^\top X$. Les composantes principales de X sont les vecteurs propres de Σ, ordonnés par valeur propre décroissante.* ■

Démonstration. Commençons par démontrer que, pour tout vecteur $\vec{w} \in \mathbb{R}^p$, la variance de la projection de X sur $\vec{w}$ vaut $w^\top \Sigma w$.

La projection de $X \in \mathbb{R}^{n\times p}$ sur $\vec{w} \in \mathbb{R}^p$ est le vecteur $\vec{z} = Xw$. Comme X est centrée, la moyenne de $\vec{z}$ vaut

$$\frac{1}{n}\sum_{i=1}^{n} z_i = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{p} x_j^i w_j = \frac{1}{n}\sum_{j=1}^{p} w_j \sum_{i=1}^{n} x_j^i = 0.$$

Sa variance vaut

$$\text{Var}[\vec{z}] = \frac{1}{n}\sum_{i=1}^{n} z z_i^2 = \frac{1}{n}\vec{w}^\top X^\top X \vec{w} = \vec{w}^\top \Sigma \vec{w}.$$

Appelons maintenant $\vec{w}_1 \in \mathbb{R}^p$ la première composante principale. $\vec{w}_1$ est orthonormé et tel que la variance de $X\vec{w}_1$ soit maximale :

$$\vec{w}_1 = \underset{\vec{w}\in\mathbb{R}^p}{\arg\max}\, \vec{w}^\top \Sigma \vec{w} \text{ avec } ||\vec{w}_1||_2 = 1. \tag{11.3}$$

Il s'agit d'un problème d'optimisation quadratique sous contrainte d'égalité (voir section A.4), que l'on peut résoudre en introduisant le multiplicateur de Lagrange $\alpha_1 > 0$ et en écrivant le lagrangien

$$L(\alpha_1, \vec{w}) = \vec{w}^\top \Sigma \vec{w} - \alpha_1 \left(||\vec{w}||_2 - 1\right).$$

Par dualité forte, le maximum de $\vec{w}^\top \Sigma \vec{w}$ sous la contrainte $||\vec{w}_1||_2 = 1$ est égal à $\min_{\alpha_1} \sup_{\vec{w}\in\mathbb{R}^p} L(\alpha_1, \vec{w})$. Le supremum du lagrangien est atteint en un point où son gradient s'annule, c'est-à-dire qui vérifie

$$2\Sigma\vec{w} - 2\alpha_1\vec{w} = 0.$$

Ainsi, $\Sigma\vec{w}_1 = \alpha_1\vec{w}_1$ et $(\alpha_1, \vec{w}_1)$ forment un couple (valeur propre, vecteur propre) de Σ.

Parmi tous les vecteurs propres de Σ, $\vec{w}_1$ est celui qui maximise la variance $\vec{w}_1^\top \Sigma \vec{w}_1 = \alpha_1 ||\vec{w}_1||_2 = \alpha_1$. Ainsi, α_1 est la plus grande valeur propre de Σ (rappelons que Σ étant définie par $X^\top X$ est semi-définie positive et que toutes ses valeurs propres sont positives.)

La deuxième composante principale de X vérifie

$$\vec{w}_2 = \underset{\vec{w}\in\mathbb{R}^p}{\arg\max}\, \vec{w}^\top \Sigma \vec{w} \text{ avec } ||\vec{w}_2||_2 = 1 \text{ et } \vec{w}^\top \vec{w}_1 = 0. \tag{11.4}$$

Cette dernière contrainte nous permet de garantir que la base des composantes principales est orthonormée.

Nous introduisons donc maintenant deux multiplicateurs de Lagrange $\alpha_2 > 0$ et $\beta_2 > 0$ et obtenons le lagrangien

$$L(\alpha_2, \beta_2, \vec{w}) = \vec{w}^{\top}\Sigma\vec{w} - \alpha_2\left(||\vec{w}||_2^2 - 1\right) - \beta_2\vec{w}^{\top}\vec{w}_1.$$

Comme précédemment, son supremum en $\vec{w}$ est atteint en un point où son gradient s'annule :

$$2\Sigma\vec{w}_2 - 2\alpha_2\vec{w}_2 - \beta_2\vec{w}_1 = 0.$$

En multipliant à gauche par $\vec{w}_1^{\top}$, on obtient

$$2\vec{w}_1^{\top}\Sigma\vec{w}_2 - 2\alpha_2\vec{w}_1^{\top}\vec{w}_2 - \beta_2\vec{w}_1^{\top}\vec{w}_1 = 0$$

d'où l'on conclut que $\beta_2 = 0$ et, en remplaçant dans l'équation précédente, que, comme pour $\vec{w}_1$, $2\Sigma\vec{w}_2 - 2\alpha_1\vec{w}_2 = 0$. Ainsi $(\alpha_2, \vec{w}_2)$ forment un couple (valeur propre, vecteur propre) de Σ et α_2 est maximale : il s'agit donc nécessairement de la deuxième valeur propre de Σ.

Le raisonnement se poursuit de la même manière pour les composantes principales suivantes. □

Remarque

Alternativement, on peut prouver le théorème 11.1 en observant que Σ, qui est par construction définie positive, est diagonalisable par un changement de base orthonormée : $\Sigma = Q^{\top}\Lambda Q$, où $\Lambda \in \mathbb{R}^{p\times p}$ est une matrice diagonale dont les valeurs diagonales sont les valeurs propres de Σ. Ainsi,

$$\vec{w}_1^{\top}\Sigma\vec{w}_1 = \vec{w}_1^{\top}Q^{\top}\Lambda Q\vec{w}_1 = (Q\vec{w}_1)^{\top}\Lambda\,(Q\vec{w}_1)\,.$$

Si l'on pose $\vec{v} = Q\vec{w}_1$, il s'agit donc de trouver $\vec{v}$ de norme 1 (Q étant orthonormée et $\vec{w}_1$ de norme 1) qui maximise $\sum_{j=1}^{p} v_j^2\lambda_j$. Comme Σ est définie positive, $\lambda_j \geq 0\ \forall j = 1, \ldots, p$. De plus, $||\vec{v}||_2 = 1$ implique que $0 \leq v_j^1 \leq 1\ \forall j = 1, \ldots, p$. Ainsi, $\sum_{j=1}^{p} v_j^2\lambda_j \leq \max_{j=1,\ldots,p}\lambda_j \sum_{j=1}^{p} v_j^2 \leq \max_{j=1,\ldots,p}\lambda_j$ et ce maximum est atteint quand $v_j = 1$ et $v_k = 0\ \forall k \neq j$. On retrouve ainsi que $\vec{w}_1$ est le vecteur propre correspondant à la plus grande valeur propre de Σ, et ainsi de suite.

Décomposition en valeurs singulières

Théorème 11.2 *Soit $X \in \mathbb{R}^{n\times p}$ une matrice centrée. Les composantes principales de X sont ses vecteurs singuliers à droite ordonnés par valeur singulière décroissante.* ■

Démonstration. Si l'on écrit X sous la forme $UDV^{\top}$ où $U \in \mathbb{R}^{n\times n}$ et $V \in \mathbb{R}^{p\times p}$ sont orthogonales, et $D \in \mathbb{R}^{n\times p}$ est diagonale, alors

$$\Sigma = X^{\top}X = VDU^{\top}UDV^{\top} = VD^2V^{\top}$$

et les valeurs singulières de X (les entrées de D) sont les racines carrées des valeurs propres de Σ, tandis que les vecteurs singuliers à droite de X (les colonnes de V) sont les vecteurs propres de Σ. □

En pratique, les implémentations de la décomposition en valeurs singulières (ou SVD) sont numériquement plus stables que celles de décomposition spectrale. On préfèrera donc calculer les composantes principales de X en calculant la SVD de X plutôt que la décomposition spectrale de $X^{\top}X$.

Choix du nombre de composantes principales

Réduire la dimension des données par une ACP implique de *choisir* un nombre de composantes principales à conserver. Pour ce faire, on utilise la *proportion de variance expliquée* par ces composantes : la variance de X s'exprime comme la trace de Σ, qui est elle-même la somme de ses valeurs propres.

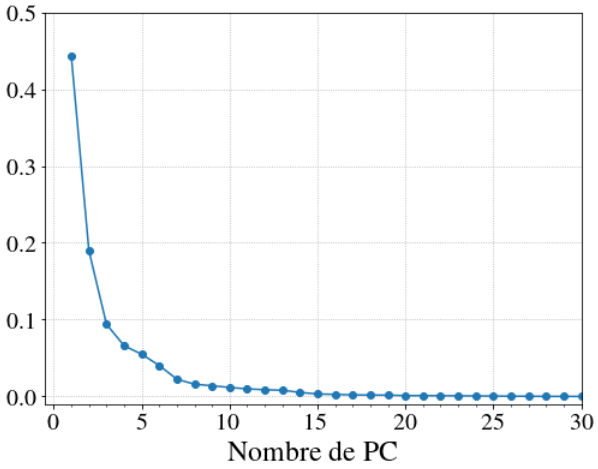
Ainsi, si l'on décide de conserver les m premières composantes principales de X, la proportion de variance qu'elles expliquent est

$$\frac{\alpha_1 + \alpha_2 + \cdots + \alpha_m}{\mathrm{Tr}(\Sigma)}$$

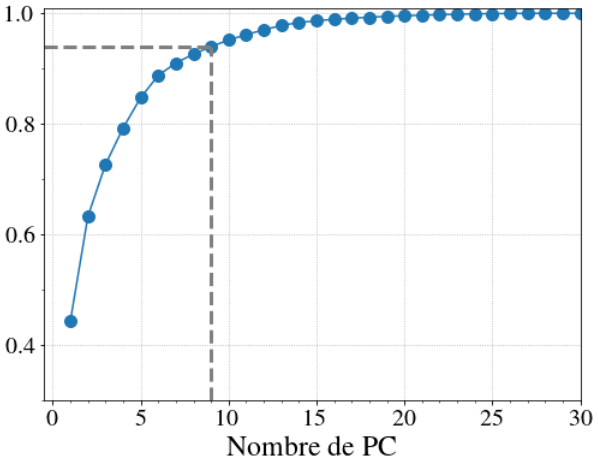
où $\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_p$ sont les valeurs propres de Σ par ordre décroissant.

Il est classique de s'intéresser à l'évolution, avec le nombre de composantes, soit de la proportion de variance expliquée par chacune d'entre elles, soit à cette proportion cumulée, que l'on peut représenter visuellement sur un *scree plot* (figure 11.4a). Ce graphe peut nous servir à déterminer :

— soit le nombre de composantes principales qui explique un pourcentage de la variance que l'on s'est initialement fixé (par exemple, sur la figure 11.4b, 95%) ;
— soit le nombre de composantes principales correspondant au « coude » du graphe, à partir duquel ajouter une nouvelle composante principale ne semble plus informatif.



(A) Pourcentage de variance expliqué par chacune des composantes principales. À partir de 6 composantes principales, ajouter de nouvelles composantes n'est plus vraiment informatif.



(B) Pourcentage cumulé de variance expliquée par chacune des composantes principales. Si on se fixe une proportion de variance expliquée de 95%, on peut se contenter de 10 composantes principales.

FIGURE 11.4 – Pour choisir le nombre de composantes principales, on utilise le pourcentage de variance expliquée.

11.3.2 Factorisation de la matrice des données

Soit un nombre m de composantes principales, calculées par une ACP, représentées par une matrice $W \in \mathbb{R}^{p \times m}$. La représentation réduite des n observations dans le nouvel espace de dimension m s'obtient en projetant X sur les colonnes de W, autrement dit en calculant

$$H = W^\top X. \tag{11.5}$$

La matrice $H \in \mathbb{R}^{m \times n}$ peut être interprétée comme une représentation *latente* (ou cachée, *hidden* en

anglais d'où la notation H) des données. C'est cette représentation que l'on a cherché à découvrir grâce à l'ACP.

Les colonnes de W étant des vecteurs orthonormés (il s'agit de vecteurs propres de $XX^\top$), on peut multiplier l'équation 11.5 à gauche par W pour obtenir une *factorisation* de X :

$$X = WH. \tag{11.6}$$

On appelle ainsi les lignes de H les *facteurs* de X.

Analyse factorielle

Cette factorisation s'inscrit dans le cadre plus général de *l'analyse factorielle*.

Supposons que les observations $\{\vec{x}^{\,1}, \vec{x}^{\,2}, \ldots, \vec{x}^{\,n}\}$ soient les réalisations d'une variable aléatoire p-dimensionnelle $\vec{x}$, générée par le modèle

$$\vec{x} = W\vec{h} + \epsilon, \tag{11.7}$$

où $\vec{h}$ est une variable aléatoire m-dimensionnelle qui est la représentation latente de $\vec{x}$, et ϵ un bruit gaussien : $\epsilon \sim \mathcal{N}(0, \Psi)$.

Supposons les données centrées en 0, et les variables latentes $\vec{h}^{\,1}, \vec{h}^{\,2}, \ldots, \vec{h}^{\,n}$ (qui sont des réalisations de $\vec{h}$) indépendantes et gaussiennes de variance unitaire, c'est-à-dire $\vec{h} \sim \mathcal{N}(0, I_m)$ où I_m est la matrice identité de dimensions $m \times m$. Alors $W\vec{h}$ est centrée en 0, et sa covariance est $WW^\top$. Alors $\vec{x} \sim \mathcal{N}(0, WW^\top + \Psi)$.

Si on considère de plus que ϵ est *isotropique*, autrement dit que $\Psi = \sigma^2 I_p$, alors $\vec{x} \sim \mathcal{N}(0, WW^\top + \sigma^2 I_p)$, on obtient ce que l'on appelle *ACP probabiliste* (Tipping et Bishop, 1999). Les paramètres W et σ de ce modèle peuvent être obtenus par maximum de vraisemblance.

L'ACP classique est un cas limite de l'ACP probabiliste, obtenu quand la covariance du bruit devient infiniment petite ($\sigma^2 \to 0$).

Plus généralement, on peut faire l'hypothèse que les variables observées $x_1, x_2, \ldots, x_p$ sont conditionnellement indépendantes étant données les variables latentes $h_1, h_2, \ldots, h_m$. Dans ce cas, Ψ est une matrice diagonale, $\Psi = \text{diag}(\psi_1, \psi_2, \ldots, \psi_p)$, où ψ_j décrit la variance spécifique à la variable x_j. Les valeurs de W, σ et $\psi_1, \psi_2, \ldots, \psi_p$ peuvent encore une fois être obtenues par maximum de vraisemblance. C'est ce que l'on appelle *l'analyse factorielle*.

Dans l'analyse factorielle, nous ne faisons plus l'hypothèse que les nouvelles dimensions sont orthogonales. En particulier, il est donc possible d'obtenir des dimensions dégénérées, autrement dit des colonnes de W dont toutes les coordonnées sont 0.

Factorisation positive de matrices

Dans le cas où les valeurs de X sont toutes positives, il peut être plus interprétable de chercher une factorisation similaire à l'équation 11.6, mais telle que W et H soient toutes les deux aussi à valeurs positives.

Dans ce cas, on parle de *factorisation positive de matrice* (ou *NMF* de l'anglais *Non-negative Matrix Factorization*). Le problème est alors formalisé comme

$$\underset{W \in \mathbb{R}_+^{p \times m}, H \in \mathbb{R}_+^{m \times n}}{\arg\min} ||X - WH||_F^2, \tag{11.8}$$

où $||.||_F$ désigne la *norme de Frobenius* d'une matrice, autrement dit la racine carrée de la somme des carrés de ses entrées. Ainsi, $||X - WH||_F^2$ compare les matrices X et de WH entrée par entrée.

Le problème 11.8 peut être résolu par un algorithme à directions de descente (voir section A.3.3).

Exemple

Cette technique peut être appliquée, par exemple, dans l'analyse des notes données par des spectateurs à différents films. Supposons avoir n spectateurs qui ont donné des notes entre 0 et 5 à p films ; une factorisation non négative permet d'interpréter une de ces notes (x_j^i) comme la combinaison de l'importance pour le spectateur i des m aspects d'un film (donnée par $\vec{h}^{\,i} \in \mathbb{R}_+^m$), et de la présence de chacun de ces aspects dans le film (donnée par la j-ème ligne de W).

Remarque

Cette technique peut aussi être utilisée pour faire de la *complétion de matrice*, c'est-à-dire dans le cas où X a des valeurs manquantes (tous les spectateurs n'ont pas noté tous les films). Plutôt que de chercher à minimiser $||X - WH||_F^2$, on restreint cette somme aux entrées connues de X : le problème 11.8 devient alors

$$\underset{W\in\mathbb{R}_+^{p\times m},H\in\mathbb{R}_+^{m\times n}}{\arg\min} \sum_{i,j \text{non manquants}} (X_{ij} - (WH)_{ij})^2 . \tag{11.9}$$

Ce problème se résout aussi par un algorithme à directions de descente (voir section A.3.3), et on peut alors approcher X par le produit WH. Cela permet de prédire les entrées manquantes de X par les entrées correspondantes de WH.

Cette technique est utilisée dans le cas du *filtrage collaboratif* (voir section 8.4), où les colonnes de X correspondent à des utilisateurs et ses lignes à des produits.

11.3.3 Auto-encodeurs

Nous avons vu à la section 7.2.5 qu'il est possible d'interpréter les réseaux de neurones artificiels comme un outil pour apprendre une représentation des données (celle issue de la dernière couche intermédiaire) qui se prête bien à un apprentissage supervisé linéaire. Cette idée a conduit à la notion d'*auto-encodeur*, qui permet d'utiliser les réseaux de neurones pour faire de la réduction de dimension non supervisée.

Définition 11.6 (Autoencodeur) On appelle *auto-encodeur* un réseau de neurones dont la couche de sortie est identique à la couche d'entrée, en passant par une couche intermédiaire de taille inférieure à celles-ci. La sortie de cette couche intermédiaire constitue alors une nouvelle représentation plus compacte des données. ■

Bien qu'il existe des auto-encodeurs linéaires, on leur préfère généralement les *machines de Boltzmann restreintes*, ou *RBM* pour *Restricted Boltzmann Machines.* Elles ont été proposées par Paul Smolensky (1986).

Une machine de Boltzmann restreinte contient deux couches de neurones, comme illustré sur la figure 11.5. La première couche contient p neurones, une par variable x_j représentant les données ; la deuxième en contient m, où $m < p$. Le vecteur $\vec{z} = (z_1, z_2, \ldots, z_m)$ en sortie de cette couche constitue une représentation réduite de $\vec{x}$. Nous supposerons ces variables binaires, à valeur dans $\{0, 1\}$: elles peuvent correspondre à la couleur d'un pixel sur une image en noir et blanc.

Cette architecture est dite *restreinte* car il n'y a pas de connexions entre neurones d'une même couche, contrairement aux machines de Boltzmann proposées précédemment et utilisées dans le cadre de la physique statistique. Par contre, les connexions entre les deux couches vont dans les deux sens : de la couche d'entrée vers la couche intermédiaire, et aussi de la couche intermédiaire vers la couche d'entrée. Les poids de connexion seront choisis de sorte à minimiser la différence entre l'entrée et la sortie, obtenue après un passage vers l'avant puis un retour vers l'arrière dans le réseau.

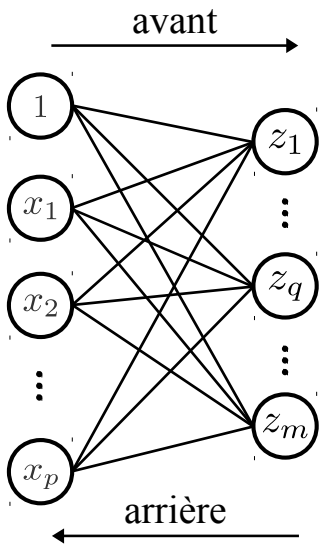


FIGURE 11.5 – Architecture d'une machine de Boltzmann restreinte qui apprend une représentation en m dimensions de données en dimension p.

La particularité des machines de Boltzmann restreintes est d'utiliser des fonctions d'activation stochastique :

$$\mathbb{P}(z_q = 1|\vec{x}) = \sigma\left(b_q + \sum_{j=1}^{p} w_{jq}x_j\right), \tag{11.10}$$

où b_q est le biais du q-ème neurone intermédiaire, et

$$\mathbb{P}(x_j = 1|\vec{z}) = \sigma\left(a_j + \sum_{q=1}^{m} w_{jq}z_q\right), \tag{11.11}$$

où a_j est le biais du j-ème neurone visible.

En s'inspirant de la physique statistique, on peut alors définir l'*énergie* de ce modèle comme

$$E(\vec{x}, \vec{z}) = -\sum_{j=1}^{p} a_j x_j - \sum_{q=1}^{m} b_h z_h - \sum_{j=1}^{p}\sum_{q=1}^{m} x_j w_{jq} z_q. \tag{11.12}$$

La loi de probabilité jointe de $\vec{x}$ et $\vec{z}$ selon ce modèle peut s'écrire comme celle d'une distribution de Boltzmann en fonction de cette énergie :

$$\mathbb{P}(\vec{x}, \vec{z}) = \frac{1}{Z} e^{-E(\vec{x},\vec{z})} \tag{11.13}$$

où la constante de normalisation Z peut être vue comme une fonction de partition et s'écrit en sommant $\mathbb{P}(\vec{x}, \vec{z})$ sur tous les états possibles :

$$Z = \sum_{\vec{u}\in\{0,1\}^p, \vec{v}\in\{0,1\}^m} e^{-E(\vec{u},\vec{v})}. \tag{11.14}$$

Ainsi, nous pouvons apprendre les coefficients $\{a_j\}_{j=1,\dots,p}$, $\{b_q\}_{q=1,\dots,m}$, et $\{w_{jq}\}_{j=1,\dots,p,q=1,\dots,m}$ par maximum de vraisemblance.

Pour une observation $\vec{x}^{\,i}$ donnée, l'opposé du log de la vraisemblance vaut :

$$-\log P(\vec{x}^{\,i}) = -\log \sum_{\vec{v}} P(\vec{x}^{\,i}, \vec{v}) = \log \sum_{\vec{u},\vec{v}} e^{-E(\vec{u},\vec{v})} - \log \sum_{\vec{v}} e^{-E(\vec{x}^{\,i},\vec{v})}. \tag{11.15}$$

Il nous faut maintenant dériver cette expression par rapport à chacun des paramètres que l'on cherche à déterminer. En notant θ un de ces paramètres (ainsi θ peut indifféremment représenter a_j, b_q, ou w_{jq}), la dérivée partielle de $-\log P(\vec{x}^{\,i})$ par rapport à θ est :

$$\begin{aligned}\frac{\partial -\log P(\vec{x}^{\,i})}{\partial \theta} &= -\frac{1}{\sum_{\vec{u},\vec{v}} e^{-E(\vec{u},\vec{v})}} \sum_{\vec{u},\vec{v}} \frac{\partial E(\vec{u},\vec{v})}{\partial \theta} e^{-E(\vec{u},\vec{v})} + \frac{1}{\sum_{\vec{v}} e^{-E(\vec{x}^{\,i},\vec{v})}} \sum_{\vec{v}} \frac{\partial E(\vec{x}^{\,i},\vec{v})}{\partial \theta} e^{-E(\vec{x}^{\,i},\vec{v})} \\ &= -\sum_{\vec{u},\vec{v}} \mathbb{P}(\vec{u},\vec{v}) \frac{\partial E(\vec{u},\vec{v})}{\partial \theta} + \sum_{\vec{v}} \mathbb{P}(\vec{v}|\vec{x}^{\,i}) \frac{\partial E(\vec{x}^{\,i},\vec{v})}{\partial \theta}\end{aligned}$$

Ce gradient se décompose donc en un *gradient négatif*,

$$-\sum_{\vec{u},\vec{v}} \mathbb{P}(\vec{u},\vec{v}) \frac{\partial E(\vec{u},\vec{v})}{\partial \theta} = -\mathbb{E}\left[\frac{\partial E(\vec{u},\vec{v})}{\partial \theta}\right] \tag{11.16}$$

et un *gradient positif*,

$$\sum_{\vec{v}} \mathbb{P}(\vec{v}|\vec{x}^{\,i}) \frac{\partial E(\vec{x}^{\,i},\vec{v})}{\partial \theta} = \mathbb{E}\left[\frac{\partial E(\vec{x}^{\,i},\vec{v})}{\partial \theta}\right]. \tag{11.17}$$

Pour approcher ces espérances, on utilise une procédure d'*échantillonnage de Gibbs*, qui consiste à itérativement tirer une observation $\vec{x}^{\,i}$, faire une passe en avant pour obtenir $\vec{z}^{\,i}$, faire une passe en arrière pour obtenir la sortie $\vec{x}^{\,\prime i}$, puis une dernière passe en avant pour obtenir $\vec{z}^{\,\prime i}$, et approcher notre gradient par

$$\frac{\partial E(\vec{x}^{\,i},\vec{z}^{\,i})}{\partial \theta} - \frac{\partial E(\vec{x}^{\,\prime i},\vec{z}^{\,\prime i})}{\partial \theta}. \tag{11.18}$$

Les dérivées partielles de l'énergie E définie par l'équation 11.12 sont simples à calculer.

On obtient ainsi l'algorithme dit de *divergence contrastive* pour entraîner une machine de Boltzmann restreinte, et proposé initialement par Geoff Hinton (2002).

Définition 11.7 (Divergence contrastive) Étant donnée une machine de Boltzmann restreinte dont l'énergie est donnée par l'équation 11.12 et une vitesse d'apprentissage η, la procédure de *divergence contrastive*, ou *contrastive divergence* en anglais, consiste à apprendre les poids a_j, b_q et w_{jq} en itérant sur les observations après avoir initialisé ces poids à une valeur aléatoire. Pour une observation $\vec{x}^{\,i}$, on effectue les étapes suivantes :

1. Tirer un vecteur intermédiaire $\vec{z}^{\,i}$ selon $\mathbb{P}(\vec{z}|\vec{x}^{\,i})$;
2. Calculer le gradient positif, qui vaut x^i_j pour $\theta = a_j$, z^i_q pour $\theta = b_q$, et $x^i_j z^i_q$ pour $\theta = w_{jq}$;
3. Tirer une reconstruction $\vec{x}^{\,\prime i}$ de $\vec{x}^{\,i}$ selon $\mathbb{P}(\vec{x}|\vec{z}^{\,i})$;
4. Tirer un vecteur intermédiaire $\vec{z}^{\,\prime i}$ selon $\mathbb{P}(\vec{z}|\vec{x}^{\,\prime i})$;
5. Calculer le gradient négatif, qui vaut $-x^{\prime i}_j$ pour $\theta = a_j$, $-z^{\prime i}_q$ pour $\theta = b_q$, et $-x^{\prime i}_j z^{\prime i}_q$ pour $\theta = w_{jq}$;
6. Mettre les poids à jour selon

$$w_{jq} \leftarrow w_{jq} + \eta\left(x^i_j z^i_q - x^{\prime i}_j z^{\prime i}_q\right); \quad a_j \leftarrow a_j + \eta\left(x^i_j - x^{\prime i}_j\right); \quad b_q \leftarrow b_q + \eta\left(z^i_q - z^{\prime i}_q\right).$$

■

Remarque

Il est possible d'empiler les RBMs pour obtenir une architecture profonde, appelée *deep belief network* (ou *DBN*). Cette architecture permet d'effectuer une réduction de dimension non supervisée, mais peut aussi être utilisée dans un cadre supervisé. Il s'agit alors de connecter la dernière couche du DBN à une couche de sortie ; on obtiendra alors un réseau de neurones supervisé. Les DBN, proposés par Geoff Hinton et Ruslan Salakhutdinov, figurent parmi les premiers réseaux de neurones profonds réalisés (Hinton et Salakhutdinov, 2006).

11.3.4 Autres approches non linéaires

De nombreuses autres approches ont été proposées pour réduire la dimension des données de manière non linéaire. Parmi elles, nous en abordons ici quelques-unes parmi les plus populaires ; les expliquer de manière détaillée dépasse le propos de cet ouvrage introductif.

Analyse en composantes principales à noyau

Nous commencerons par noter que l'analyse en composantes principales se prête à l'utilisation de l'astuce du noyau (cf. section 10.3.3). La méthode qui en résulte est appelée *kernel PCA*, ou *kPCA*.

Positionnement multidimensionnel

Le *positionnement multidimensionnel*, ou *multidimensional scaling* (*MDS*) (Cox et Cox, 1994), se base sur une matrice de *dissimilarité* $D \in \mathbb{R}^{n\times n}$ entre les observations : il peut s'agir d'une distance métrique, mais ce n'est pas nécessaire. Le but de l'algorithme est alors de trouver une représentation des données qui préserve cette dissimilarité :

$$X^* = \underset{Z\in\mathbb{R}^{n\times m}}{\arg\min} \sum_{i=1}^{n}\sum_{l=i+1}^{n} \left(\left|\left|\vec{z}^{\,i} - \vec{z}^{\,l}\right|\right|_2 - D_{il}\right)^2. \tag{11.19}$$

Si l'on utilise la distance euclidienne comme dissimilarité, alors le MDS est équivalent à une ACP.

Le positionnement multidimensionnel peut aussi s'utiliser pour positionner dans un espace de dimension m des points dont on ne connaît pas les coordonnées. Il s'applique par exemple très bien à repositionner des villes sur une carte à partir uniquement des distances entre ces villes.

Une des limitations de MDS est de ne chercher à conserver la distance entre les observations que globalement. Une façon efficace de construire la matrice de dissimilarité de MDS de sorte à conserver la structure locale des données est l'algorithme *IsoMap* (Tenenbaum et al., 2000). Il s'agit de construire un graphe de voisinage entre les observations en reliant chacune d'entre elles à ses k plus proches observations voisines. Ces arêtes peuvent être pondérée par la distance entre les observations qu'elles relient. Une dissimilarité entre observations peut ensuite être calculée sur ce graphe de voisinage, par exemple via la longueur du plus court chemin entre deux points.

t-SNE

Enfin, l'algorithme *t-SNE*, pour *t-Student Neighborhood Embedding*, proposé en 2008 par Laurens van der Maaten and Geoff Hinton, propose d'approcher la distribution des distances entre observations par une loi de Student (van der Maaten et Hinton, 2008). Pour chaque observation $\vec{x}^{\,i}$, on définit P_i comme la loi de probabilité définie par

$$P_i(\vec{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left|\left|\vec{x} - \vec{x}^{\,i}\right|\right|_2^2}{2\sigma^2}\right). \tag{11.20}$$

t-SNE consiste alors à résoudre

$$\arg\min_{Q} \sum_{i=1}^{n} KL(P_i||Q_i) \tag{11.21}$$

où KL dénote la divergence de Kullback-Leibler (voir section 2.4.1) et Q est choisie parmi les distributions de Student de dimension inférieure à p. Attention, cet algorithme trouve un minimum local et non global, et on pourra donc obtenir des résultats différents en fonction de son initialisation. De plus, sa complexité est quadratique en le nombre d'observations.

Points clefs

— Réduire la dimension des données avant d'utiliser un algorithme d'apprentissage supervisé permet d'améliorer ses besoins en temps et en espace, mais aussi ses performances.

— On distingue la sélection de variables, qui consiste à éliminer des variables redondantes ou peu informatives, de l'extraction de variable, qui consiste à générer une nouvelle représentation des données.

— Projeter les données sur un espace de dimension 2 grâce à, par exemple, une ACP ou t-SNE, permet de les visualiser.

— De nombreuses méthodes permettent de réduire la dimension des variables.

Pour aller plus loin

- Le tutoriel de Shlens (2014) est une introduction détaillée à l'analyse en composantes principales.
- Pour une revue des méthodes de sélection de variables, on pourra se référer à Guyon et Elisseeff (2003).
- Pour plus de détails sur la NMF, on pourra par exemple se tourner vers Lee et Seung (1999)
- Pour plus de détails sur les méthodes des sélection de sous-ensemble de variables (wrapper methods), on pourra se référer à l'ouvrage de Miller (1990)
- Une page web est dédiée à Isomap : `http://web.mit.edu/cocosci/isomap/isomap.html`.
- Pour plus de détails sur l'utilisation de t-SNE, on pourra se référer à la page `https://lvdmaaten.github.io/tsne/` ou à la publication interactive de Wattenberg et al. (2016).

Bibliographie

Cox, T. F. et Cox, M. A. A. (1994). *Multidimensional Scaling*. Chapman and Hall., London.

Guyon, I. et Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3 :1157–1182.

Hinton, G. E. (2002). Training product of experts by minimizing contrastive divergence. *Neural Computation*, 14 :1771–1800.

Hinton, G. E. et Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313 :504–507.

Kozachenko, L. F. et Leonenko, N. N. (1987). A statistical estimate for the entropy of a random vector. *Problemy Peredachi Informatsii*, 23 :9–16.

Lee, D. D. et Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755) :788–791.

Miller, A. J. (1990). *Subset Selection in Regression*. Chapman and Hall., London.

Shlens, J. (2014). A Tutorial on Principal Component Analysis. *arXiv [cs, stat]*. arXiv : 1404.1100.

Smolensky, P. (1986). Information processing in dynamical systems : foundations of harmony theory. In *Parallel Distributed Processing : Explorations in the Microstructure of Cognition*, volume 1 : Foundations, chapter 6, pages 194–281. MIT Press, Cambridge, MA.

Tenenbaum, J. B., de Silva, V., et Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500) :2319–2323.

Tipping, M. E. et Bishop, C. M. (1999). Probabilistic principal components analysis. *Journal of the Royal Statistical Society Series B*, 61 :611–622.

van der Maaten, L. et Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9 :2579–2605.

Wattenberg, M., Viégas, F., et Johnson, I. (2016). How to use t-SNE effectively. *Distill*. `http://distill.pub/2016/misread-tsne`.

Chapitre 12

Clustering

Comment étudier des données non étiquetées ? La dimension de réduction nous permet de les visualiser ; mais les méthodes dites de *partitionnement de données*, ou *clustering*, nous permettent d'aller beaucoup plus loin. Il s'agit de séparer les données en sous-groupes homogènes, appelés *clusters*, qui partagent des caractéristiques communes. Dans ce chapitre, nous détaillerons l'intérêt de ces techniques et verrons trois types d'approches de clustering : le clustering hiérarchique, le clustering par centroïdes, et le clustering par densité.

Objectifs

- Expliquer l'intérêt d'un algorithme de clustering
- Évaluer le résultat d'un algorithme de clustering
- Implémenter un clustering hiérarchique, un clustering par la méthode des k moyennes, et un clustering par densité.

12.1 Pourquoi partitionner ses données

Les algorithmes de partitionnement de données permettent d'effectuer une analyse exploratoire sur des données non étiquetées. Ils permettent par exemple d'identifier des utilisateurs qui ont des comportements similaires (ce que l'on appelle la *segmentation de marché*), des communautés sur un réseau social, des motifs récurrents dans des transactions financières, des pixels d'un même objet dans une image (*segmentation d'image*) ou des patients dont la maladie s'explique par un même profil génétique.

Ils permettent aussi de visualiser les données, en se contentant de regarder un exemple représentatif par cluster.

Enfin, les algorithmes de clustering permettent de transférer à toutes les observations du même cluster les propriétés que l'on sait vraies de l'un des éléments de ce cluster. Cela est particulièrement utile dans le cas où l'étiquetage des données est difficile ou coûteux.

Exemple

Prenons l'exemple de l'annotation d'un corpus de documents texte. Annoter manuellement chacun de ces documents par le ou les sujets qu'il couvre est un travail fastidieux. Les personnes qui l'effectuent sont d'ailleurs susceptibles de commettre involontairement des erreurs d'inattention. Il est ainsi moins coûteux, voire plus efficace, d'utiliser un algorithme de clustering pour regrouper automatiquement ces documents par sujet. Il suffira alors d'avoir recours à un intervenant humain pour assigner un sujet à chaque cluster en lisant uniquement un ou deux des documents qu'il contient.

12.2 Évaluer la qualité d'un algorithme de clustering

Le clustering n'étant pas supervisé, il nous faut mettre au point des critères d'évaluation qui ne dépendent pas d'une vérité terrain (c'est-à-dire d'étiquettes connues). La tâche est plus délicate que dans le cadre de l'apprentissage supervisé, dans lequel le but à atteindre est beaucoup plus clair. Cependant, elle n'est pas impossible, et il existe un large éventail de mesures de la performance d'un algorithme de partitionnement des données.

Par la suite, nous supposons avoir partitionné un jeu de données non étiqueté $\mathcal{D} = \{\vec{x}^{\,1}, \vec{x}^{\,2}, \ldots, \vec{x}^{\,n}\}$ de n points d'un espace $\mathcal{X}$ en K clusters $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_K$. d est une distance sur $\mathcal{X}$ et $k(\vec{x})$ est l'indice du cluster auquel $\vec{x}$ a été assigné.

12.2.1 La forme des clusters

Pour évaluer la qualité des clusters obtenus par un algorithme de partitionnement, il est possible de s'appuyer sur le souhait formulé de regrouper entre elles les observations similaires. Ainsi, les observations appartenant à un même cluster doivent être proches, tandis que les observations dissimilaires doivent appartenir à des clusters différents.

Pour quantifier ces notions, nous allons avoir besoin de celle de *centroïde*, qui est le barycentre d'un cluster.

Définition 12.1 (Centroïde et médoïde) On appelle *centroïde* du cluster $\mathcal{C}$ le point défini par

$$\vec{\mu}_{\mathcal{C}} = \frac{1}{|\mathcal{C}|} \sum_{\vec{x} \in \mathcal{C}} \vec{x}.$$

Le *médoïde* est le point du cluster le plus proche du *centroïde* (il peut ne pas être unique, auquel cas il sera choisi arbitrairement). Il sert de représentant du cluster :

$$\vec{m}_{\mathcal{C}} = \arg\min_{\vec{x} \in \mathcal{C}} d(\vec{x}, \vec{\mu}_{\mathcal{C}}).$$

■

Que des observations proches appartiennent au même cluster peut se traduire par la notion d'homogénéité, illustrée sur la figure 12.1 :

Définition 12.2 (Homogénéité) On appelle *homogénéité* du cluster $\mathcal{C}_k$, ou *tightness* en anglais, la moyenne des distances des observations de ce cluster à son centroïde :

$$T_k = \frac{1}{|\mathcal{C}_k|} \sum_{\vec{x} \in \mathcal{C}_k} d(\vec{x}, \vec{\mu}_k).$$

Ici $\vec{\mu}_k$ est le centroïde de $\mathcal{C}_k$.

L'*homogénéité globale* d'un clustering de $\mathcal{D}$ se calcule comme la moyenne des homogénéités des clusters :

$$T = \frac{1}{K} \sum_{k=1}^{K} T_k.$$

■

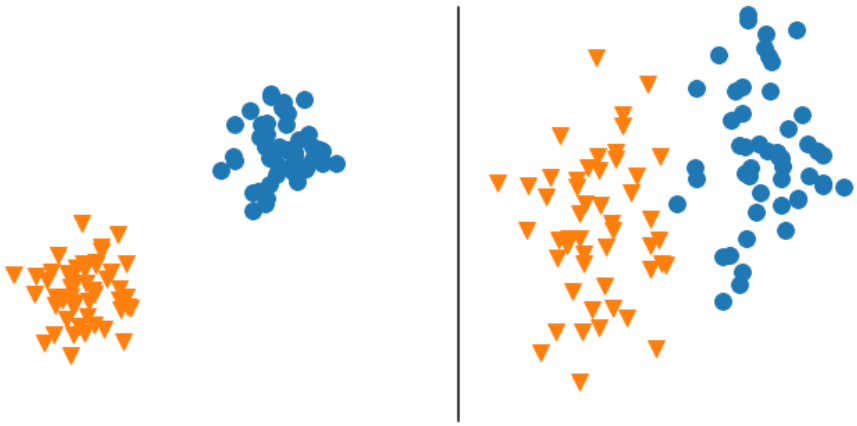


FIGURE 12.1 – Les deux clusters représentés sur le panneau de gauche sont homogènes, resserrés sur eux-mêmes : ils sont composés de points proches les uns des autres. À l'inverse, les deux clusters représentés sur le panneau de droite sont moins homogènes.

Pour quantifier à quel point les clusters sont distants les uns des autres, nous pouvons utiliser le critère de *séparabilité*, illustré sur la figure 12.2.

Définition 12.3 (Séparabilité) On appelle *séparabilité* des clusters $\mathcal{C}_k$ et $\mathcal{C}_l$ la distance entre leurs centroïdes :

$$S_{kl} = d(\vec{\mu}_k, \vec{\mu}_l).$$

La *séparabilité globale* d'un clustering de $\mathcal{D}$ se calcule comme la moyenne des séparabilités des clusters deux à deux :

$$S = \frac{2}{K(K-1)} \sum_{k=1}^{K} \sum_{k=k+1}^{K} S_{kl}.$$

■

Plutôt que de considérer les deux critères de séparabilité (que l'on souhaite élevée) et d'homogénéité (que l'on souhaite faible) séparément, il est possible de les comparer l'un à l'autre grâce à l'indice de Davies-Bouldin.

Définition 12.4 (Indice de Davies-Bouldin) On appelle *indice de Davies-Bouldin* du cluster $\mathcal{C}_k$ la valeur

$$D_k = \max_{l \neq k} \frac{T_k + T_l}{S_{kl}}.$$

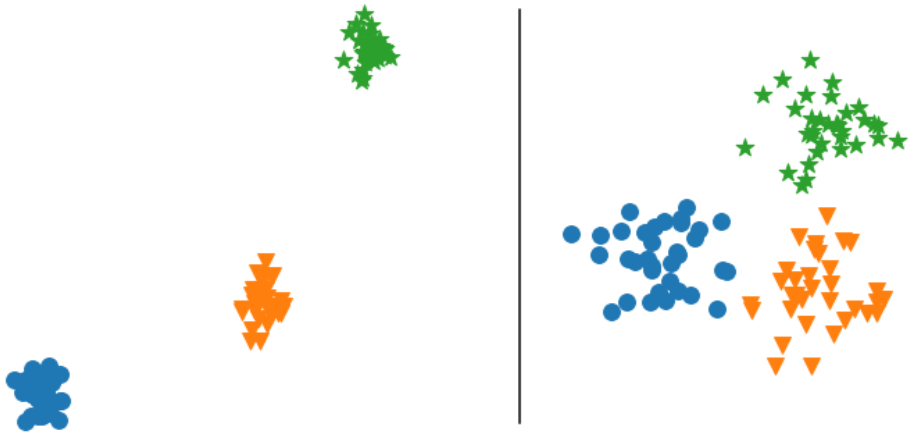


FIGURE 12.2 – Les trois clusters représentés sur le panneau de gauche sont bien séparés, contrairement à ceux représentés sur le panneau de droite qui sont proches les uns des autres.

L'*indice de Davies-Bouldin* global d'un clustering de $\mathcal{D}$ se calcule comme la moyenne des indices de Davies-Bouldin des clusters :

$$D = \frac{1}{C}\sum_{k=1}^{K} D_k.$$

■

Une autre façon de prendre en compte séparabilité et homogénéité est de calculer le *coefficient de silhouette*, qui permet de quantifier pour chacune des observations si elle appartient ou non au bon cluster.

Définition 12.5 (Coefficient de silhouette) On appelle *coefficient de silhouette* de l'observation $\vec{x} \in \mathcal{D}$ la valeur

$$s(\vec{x}) = \frac{b(\vec{x}) - a(\vec{x})}{\max(a(\vec{x}), b(\vec{x}))}$$

où $a(\vec{x})$ est la distance moyenne de $\vec{x}$ à tous les autres éléments du cluster auquel il appartient et $b(\vec{x})$ est la plus petite valeur que pourrait prendre $a(\vec{x})$, si a appartenait à un autre cluster :

$$a(\vec{x}) = \frac{1}{|\mathcal{C}_{k(\vec{x})}| - 1}\sum_{\vec{u}\in\mathcal{C}_{k(\vec{x})}, \vec{u}\neq\vec{x}} d(\vec{u},\vec{x}); \qquad b(\vec{x}) = \min_{l\neq k(\vec{x})} \frac{1}{|\mathcal{C}_l|}\sum_{\vec{u}\in\mathcal{C}_l} d(\vec{u},\vec{x}).$$

Le *coefficient de silhouette global* du clustering est son coefficient de silhouette moyen :

$$s = \frac{1}{n}\sum_{i=1}^{n} s(\vec{x}^{\,i}).$$

■

Le coefficient de silhouette de $\vec{x}$ est d'autant plus proche de 1 que son assignation au cluster $\mathcal{C}_{k(\vec{x})}$ est satisfaisante.

12.2.2 La stabilité des clusters

Un autre critère important est celui de la *stabilité* des clusters. On s'attend en effet à obtenir les mêmes clusters si on supprime ou perturbe quelques observations, ou en initialisant différemment l'algorithme de partitionnement.

Ce critère peut être utilisé pour choisir les hyperparamètres de l'algorithme : si on obtient des clusters très différents pour différentes initialisations de l'algorithme de partitionnement, cela peut indiquer que les hyperparamètres sont mal choisis.

12.2.3 Les connaissances expert

Enfin, il arrive parfois que l'on dispose d'un jeu de données partiellement étiqueté par des classes que nous aimerions retrouver par clustering. Cela peut être le cas d'un corpus de documents dont un sous-ensemble est étiqueté par sujet, ou d'une base de données d'images dont un sous-ensemble est étiqueté en fonction de ce qu'elles représentent.

On peut alors évaluer le résultat d'un algorithme de partitionnement comme on évaluerait celui d'un algorithme de classification multi-classe. Attention, il y a cependant une différence : dans le cas du clustering, peu importe que les objets de la classe k se retrouvent dans le premier, le deuxième ou le k-ème cluster. Il faut donc évaluer la concordance de la partition des données par l'algorithme de clustering avec celle induite par les étiquettes. C'est ce que permet de faire l'*indice de Rand*.

Nous supposons maintenant les observations $\vec{x}^{\,i}$ étiquetées par $y^i \in \{1, 2, \ldots, C\}$.

Définition 12.6 (Indice de Rand) On appelle *indice de Rand* la proportion de paires d'observations qui sont soit de même classe et dans le même cluster, soit de classe différente et dans deux clusters différents :

$$\mathrm{RI} = \frac{2}{n(n-1)} \sum_{i=1}^{n} \sum_{l=i+1}^{n} \delta\left(k(\vec{x}^{\,i}) = k(\vec{x}^{\,l})\right) \delta(y^i = y^l) + \delta\left(k(\vec{x}^{\,i}) \neq k(\vec{x}^{\,l})\right) \delta(y^i \neq y^l).$$

■

En bioinformatique, où il est souvent délicat d'étiqueter les objets d'étude par manque de connaissances, il est fréquent d'évaluer un clustering en le comparant à une *ontologie*, c'est-à-dire une classification d'objets (par exemple, des gènes) en catégories décrites par un vocabulaire commun et organisées de manière hiérarchique. On peut évaluer la cohérence d'un clustering avec une ontologie par une *analyse d'enrichissement*, qui consiste à évaluer s'il y a plus d'objets d'une catégorie de l'ontologie au sein d'un cluster que ce à quoi on pourrait s'attendre par hasard. Si l'on suppose les données issues d'une distribution hypergéométrique, il s'agit alors de calculer, pour un cluster $\mathcal{C}_k$, une catégorie $\mathcal{G}$, et un seuil $t \in \mathbb{N}$,

$$\mathbb{P}[|\mathcal{G} \cap \mathcal{C}_k| \geq t] = 1 - \sum_{s=1}^{t} \frac{\binom{|\mathcal{G}|}{s}\binom{n-|\mathcal{G}|}{|\mathcal{C}_k|-s}}{\binom{n}{|\mathcal{C}_k|}}. \tag{12.1}$$

En effet, le terme sous la somme est la probabilité que, lorsqu'on tire $|\mathcal{C}_k|$ éléments parmi n, s d'entre eux appartiennent à $\mathcal{G}$.

Muni de ces différentes façons d'évaluer un algorithme de partitionnement de données, nous pouvons maintenant découvrir ces algorithmes eux-mêmes. Ces algorithmes cherchent à optimiser les critères d'homogénéité et de séparabilité que nous venons de définir. Comme il n'est pas possible de le faire de manière exacte, il s'agit de le faire de manière approchée. Nous nous concentrerons dans ce chapitre sur les trois principales familles d'algorithmes de clustering : clustering hiérarchique, clustering par centroïdes, et clustering par densité.

12.3 Clustering hiérarchique

Le *clustering hiérarchique* forme des clusters séparés par récurrence : il s'agit de partitionner les données pour toutes les échelles possibles de taille de partition, dans une hiérarchie à plusieurs niveaux.

12.3.1 Dendrogramme

Le résultat d'un clustering hiérarchique peut se visualiser sous la forme d'un *dendrogramme*. Il s'agit d'un arbre dont les n feuilles correspondent chacune à une observation. Chaque nœud de l'arbre correspond à un cluster :

— la racine est un cluster contenant toutes les observations
— chaque feuille est un cluster contenant une observation
— les clusters ayant le même parent sont agglomérés en un seul cluster au niveau au-dessus
— un cluster est subdivisé en ses enfants au niveau au-dessous.

Ce sont donc les nœuds intermédiaires qui nous intéresseront le plus.

Enfin, la longueur d'une branche de l'arbre est proportionnelle à la distance entre les deux clusters qu'elle connecte.

La figure 12.3 représente un exemple de dendrogramme. Dans le cas où n est trop grand pour représenter l'intégralité de l'arbre, il est classique de le couper et de n'en représenter que la partie de la racine à un niveau que l'on choisit.

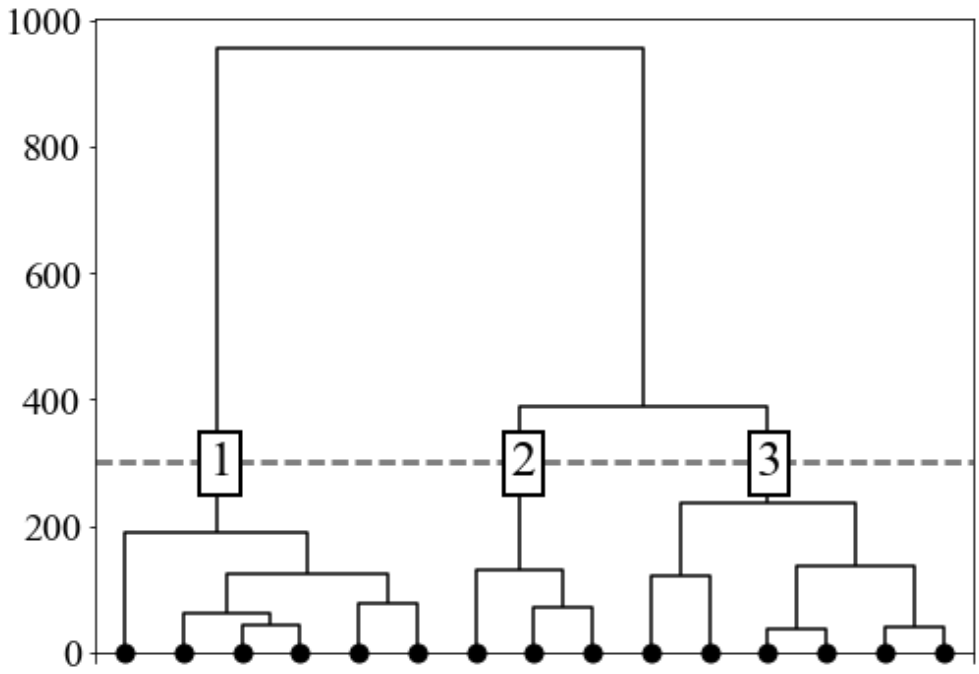


FIGURE 12.3 – Un exemple de dendrogramme. En coupant au niveau de la ligne en pointillés, on obtient 3 clusters. Chaque feuille de l'arbre (sur l'axe des abscisses) correspond à une observation.

Cette facilité à représenter visuellement le résultat d'un algorithme de clustering hiérarchique fait qu'il est très utilisé dans des domaines comme la bioinformatique.

12.3.2 Construction agglomérative ou divisive

Un clustering hiérarchique peut se construire de manière *agglomérative* ou *divisive*.

Le *clustering agglomératif*, ou *bottom-up clustering*, commence par considérer les feuilles du dendrogramme : initialement, chaque observation forme un cluster de taille 1. À chaque itération de l'algorithme, on trouve les deux clusters les plus proches, et on les agglomère en un seul cluster, et ce jusqu'à ne plus avoir qu'un unique cluster contenant les n observations.

Le *clustering divisif*, ou *top-down clustering*, est l'approche inverse. On l'initialise en considérant un seul cluster, la racine du dendrogramme, contenant toutes les observations. À chaque itération, on sépare un cluster en deux, jusqu'à ce que chaque cluster ne contienne plus qu'une seule observation.

Par la suite, nous nous concentrerons sur l'approche agglomérative.

12.3.3 Fonctions de lien

Déterminer les deux clusters les plus proches afin de les agglomérer requiert de définir une distance entre clusters ; c'est ce qu'on appelle dans le cadre du clustering une *fonction de lien*, ou *linkage* en anglais. Plusieurs approches sont possibles, qui reposent toutes sur une distance d sur $\mathcal{X}$.

Tout d'abord, on peut choisir d'agglomérer deux clusters si deux de leurs éléments sont proches. On utilise alors le *lien simple*.

Définition 12.7 (Lien simple) On appelle *lien simple*, ou *single linkage*, la distance entre deux clusters définie par

$$d_{\text{simple}}(\mathcal{C}_k, \mathcal{C}_l) = \min_{(\vec{u},\vec{v})\in\mathcal{C}_k\times\mathcal{C}_l} d(\vec{u},\vec{v}).$$

■

On peut aussi choisir d'agglomérer deux clusters si *tous* leurs éléments sont proches. On utilise alors le *lien complet*, qui est la distance maximale entre un élément du premier cluster et un élément du deuxième.

Définition 12.8 (Lien complet) On appelle *lien complet*, ou *complete linkage*, la distance entre deux clusters définie par

$$d_{\text{complet}}(\mathcal{C}_k, \mathcal{C}_l) = \max_{(\vec{u},\vec{v})\in\mathcal{C}_k\times\mathcal{C}_l} d(\vec{u},\vec{v}).$$

■

Une approche intermédiaire consiste à considérer la distance moyenne entre un élément du premier cluster et un élément du deuxième. C'est le *lien moyen*.

Définition 12.9 (Lien moyen) On appelle *lien moyen*, ou *average linkage*, la distance entre deux clusters définie par

$$d_{\text{moyen}}(\mathcal{C}_k, \mathcal{C}_l) = \frac{1}{|\mathcal{C}_k|}\frac{1}{|\mathcal{C}_l|}\sum_{\vec{u}\in\mathcal{C}_k}\sum_{\vec{v}\in\mathcal{C}_l} d(\vec{u},\vec{v}).$$

Cette distance est aussi parfois appelée *UPGMA* pour *Unweighted Paired Group Method with Arithmetic mean.* ■

Une alternative au lien moyen est le *lien centroïdal*, qui considère la distance entre les centroïdes des clusters.

Définition 12.10 (Lien centroïdal) On appelle *lien centroïdal*, ou *centroid linkage*, la distance entre deux clusters définie par

$$d_{\text{centroidal}}(\mathcal{C}_k, \mathcal{C}_l) = d\left(\frac{1}{|\mathcal{C}_k|}\sum_{\vec{u}\in\mathcal{C}_k}\vec{u}, \frac{1}{|\mathcal{C}_l|}\sum_{\vec{v}\in\mathcal{C}_l}\vec{v}\right) = d(\vec{\mu}_k, \vec{\mu}_l).$$

Cette distance est aussi parfois appelée *UPGMC* pour *Unweighted Paired Group Method with Centroid.* ■

Les fonctions de lien ci-dessus s'attachent à garantir la *séparabilité* des clusters. À l'inverse, il est possible de chercher à maximiser leur *homogénéité* : il s'agit de minimiser $T_k = \frac{1}{|\mathcal{C}_k|}\sum_{\vec{x}\in\mathcal{C}_k} d(\vec{x}, \vec{\mu}_k)$. Dans le cas où d est la distance euclidienne, alors $T_k = \frac{1}{|\mathcal{C}_k|}\sum_{\vec{x}\in\mathcal{C}_k} ||\vec{x} - \vec{\mu}_k||_2$. Le *clustering de Ward* utilise une formulation similaire : il s'agit d'agglomérer deux clusters de sorte à minimiser la variance intra-cluster du résultat.

Définition 12.11 (Inertie) On appelle *variance intra-cluster*, ou *inertie* du cluster $\mathcal{C}$ la valeur

$$\text{Var}_{\text{in}}(\mathcal{C}) = \frac{1}{|\mathcal{C}|}\sum_{\vec{x}\in\mathcal{C}} ||\vec{x} - \vec{\mu}||_2^2 .$$

L'*inertie globale* d'un clustering de $\mathcal{D}$ est alors donnée par la somme des inerties des clusters :

$$V = \sum_{k=1}^{K}\frac{1}{|\mathcal{C}_k|}\sum_{\vec{x}\in\mathcal{C}_k} ||\vec{x} - \vec{\mu}_k||_2^2 .$$

■

12.3.4 Choix du nombre de clusters

Le clustering hiérarchique a l'avantage de ne pas requérir de définir à l'avance le nombre de clusters, ce qui permet d'explorer toutes les possibilités le long du dendrogramme. Cependant, il faut généralement prendre cette décision à un moment.

Il est possible pour cela d'utiliser un dendrogramme, pour y déceler un « niveau » auquel les clusters sont clairement distants les uns des autres – on se rappellera que la longueur d'une branche est proportionnelle à la distance entre les deux clusters qu'elle sépare.

Une solution alternative est d'évaluer les différentes partitions trouvées, c'est-à-dire les différents nœuds du dendrogramme, à l'aide d'une mesure de performance telle que le coefficient de silhouette.

12.3.5 Complexité algorithmique

La complexité algorithmique du clustering hiérarchique est élevée : à chaque itération, pour décider quels clusters regrouper, il nous faudra calculer les distances deux à deux entre toutes les paires d'observations du jeu de données, pour une complexité en $\mathcal{O}(pn^2)$. Une alternative est de stocker ces distances en mémoire pour pouvoir les réutiliser, ce qui a une complexité quadratique en le nombre d'observations.

Le clustering hiérarchique est donc plus adapté aux jeux de données contenant peu d'échantillons.

12.4 Méthode des k-moyennes

Plutôt que d'explorer toutes les partitions possibles à différentes échelles, il peut être préférable de se fixer un nombre K de clusters, ce qui permet de réduire les temps de calculs.

Nous avons vu ci-dessus que le clustering de Ward cherche à minimiser la variance intra-cluster afin de créer des clusters homogènes. La méthode dite des *k-moyennes*, ou du *k-means*, proposée par Hugo Steinhaus (1957), cherche à résoudre ce problème pour un nombre de clusters K fixé. Il s'agit alors de trouver l'affectation des observations à K clusters qui minimise la variance intra-cluster globale :

$$\underset{\mathcal{C}_1,\mathcal{C}_2,\ldots,\mathcal{C}_K}{\arg\min} \sum_{k=1}^{K} \sum_{\vec{x}\in\mathcal{C}_k} ||\vec{x} - \vec{\mu}_k||_2^2 \,. \tag{12.2}$$

Cependant, résoudre ce problème de manière exacte n'est pas possible. On utilise donc une heuristique, l'*algorithme de Lloyd*, proposé par Stuart Lloyd (1982).

12.4.1 Algorithme de Lloyd

Définition 12.12 (Algorithme de Lloyd) Étant données n observations dans $\mathbb{R}^p$ et un nombre K de clusters, l'*algorithme de Lloyd* procède de la manière suivante :

1. Choisir K observations $\vec{\mu}_1, \vec{\mu}_2, \ldots, \vec{\mu}_K$ parmi les n observations, pour servir de centroïdes initiaux ;
2. Affecter chaque observation $\vec{x}^{\,i} \in \mathcal{D}$ au centroïde dont elle est le plus proche :
$$k(\vec{x}^{\,i}) = \underset{k=1,\ldots,K}{\arg\min} \left|\left|\vec{x}^{\,i} - \vec{\mu}_k\right|\right|_2 ;$$
3. Recalculer les centroïdes de chaque cluster :
$$\vec{\mu}_k = \frac{1}{|\mathcal{C}_k|} \sum_{\vec{x}^{\,i}\in\mathcal{C}_k} \vec{x}^{\,i};$$
4. Répéter les opérations 2–3 jusqu'à convergence, c'est-à-dire jusqu'à ce que les affectations ne changent plus.

■

L'algorithme de Lloyd implémente une stratégie gloutonne, et s'il converge en général très rapidement, il peut tomber dans un minimum local. Il peut donc être pertinent de le faire tourner plusieurs fois, et de garder la solution qui a la plus faible variance intra-cluster.

Si l'on doit itérer t fois l'algorithme de Lloyd, sachant que le coût de calculer Kn distances en p dimensions est de l'ordre de $\mathcal{O}(npK)$, la complexité algorithmique de l'algorithme de Lloyd est en $\mathcal{O}(npKt)$. K et t sont typiquement négligeables devant n, et cet algorithme est donc *linéaire* en le nombre d'observations, par opposition au clustering hiérarchique dont le coût est *quadratique* en n : nous avons remplacé le calcul des distances d'une observation $\vec{x}^{\,i}$ aux $n-1$ autres points du jeu de données par un calcul de sa distance à K centroïdes.

12.4.2 Forme des clusters

D'après la formulation de l'algorithme des k-moyennes (équation 12.2), chaque cluster $\mathcal{C}_k$ est constitué des observations de $\mathcal{D}$ qui sont les plus proches du centroïde $\vec{\mu}_k$. Ils forment donc un diagramme de Voronoï (cf. section 8.1.2). En particulier, cela signifie que les clusters trouvés par l'algorithme des k-moyennes sont nécessairement convexes.

Données aberrantes

L'algorithme du k-means est sensible aux données aberrantes : elles vont en effet tirer un cluster à elle. Si une observation $\vec{x}^{\,i}$ est très éloignée des autres observations, alors elle se retrouvera dans son propre cluster, tandis que le reste des données sera partitionné en $K-1$ clusters.

Cette propriété est néanmoins intéressante, car elle permet d'utiliser l'algorithme des k-moyennes justement pour détecter les observations aberrantes : ce sont celles qui sont seules dans leur cluster.

12.4.3 Variantes

k-means++

L'algorithme du k-means est stochastique : on peut obtenir des résultats différents selon l'initialisation, et certains de ces résultats peuvent avoir une inertie bien plus grande que la solution optimale.

Pour éviter ce problème, l'algorithme *k-means++* commence par initialiser les centroïdes de manière à les disperser au maximum parmi les données. Plus précisément, la procédure consiste à

1. Choisir un premier centroïde $\vec{u}^{\,1}$ aléatoirement parmi les observations $\mathcal{D}$
2. Pour $k = 2, \dots, K$:
   Choisir le k-ème centroïde $\vec{u}^{\,k}$ parmi $\mathcal{D} \setminus \vec{u}^{\,k-1}$, en suivant une loi proportionnelle au carré de la distance à $\vec{u}^{\,k-1}$, c'est-à-dire que $\vec{u}^{\,k}$ aura de fortes chances d'être éloigné de $\vec{u}^{\,k-1}$.

Cette approche ne rend pas le k-means déterministe, mais permet d'éviter les « pires » solutions.

Méthode des k-moyennes avec noyau

La méthode des k-moyennes requiert de décrire les données dans un espace euclidien, et ne peut former que des clusters convexes, ce qui peut être limitant. Cependant, l'*astuce du noyau* (cf. section 10.3.3) s'applique à l'algorithme de Lloyd.

Démonstration. Soit $\kappa : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ un noyau. Il existe un espace de Hilbert $\mathcal{H}$ et une application $\phi : \mathcal{X} \to \mathcal{H}$ telle que, pour tout $\vec{x}, \vec{x}' \in \mathcal{X} \times \mathcal{X}$, $\kappa(\vec{x}, \vec{x}') = \langle \phi(\vec{x}), \phi(\vec{x}') \rangle_{\mathcal{H}}$.

Pour appliquer l'algorithme de Lloyd aux images $\{\phi(\vec{x}^{\,1}), \phi(\vec{x}^{\,2}), \dots, \phi(\vec{x}^{\,n})\}$ des éléments de $\mathcal{D}$ dans $\mathcal{H}$, nous avons besoin de calculer, à chaque itération, la distance de $\phi(\vec{x}^{\,i})$ à chacun des K centroïdes $\vec{h}_1, \vec{h}_2, \dots, \vec{h}_K$.

La position d'un centroïde $\vec{h}_k$ se calcule comme la moyenne des images des observations appartenant à $\mathcal{C}_k$:

$$\vec{h}_k = \frac{1}{|\mathcal{C}_k|} \sum_{\vec{x}^{\,i} \in \mathcal{C}_k} \phi(\vec{x}^{\,i}).$$

La distance de l'image d'une observation $\vec{x}^{\,i}$ à un centroïde se calcule alors comme

$$\begin{aligned}
\left|\left| \phi(\vec{x}^{\,i}) - \vec{h}_k \right|\right|_2^2 &= \left|\left| \phi(\vec{x}^{\,i}) - \frac{1}{|\mathcal{C}_k|} \sum_{\vec{u} \in \mathcal{C}_k} \phi(\vec{u}) \right|\right|_2^2 \\
&= \left\langle \phi(\vec{x}^{\,i}) - \frac{1}{|\mathcal{C}_k|} \sum_{\vec{u} \in \mathcal{C}_k} \phi(\vec{u}),\ \phi(\vec{x}^{\,i}) - \frac{1}{|\mathcal{C}_k|} \sum_{\vec{u} \in \mathcal{C}_k} \phi(\vec{u}) \right\rangle_{\mathcal{H}} \\
&= \kappa(\vec{x}^{\,i}, \vec{x}^{\,i}) - \frac{2}{|\mathcal{C}_k|} \sum_{\vec{u} \in \mathcal{C}_k} \kappa(\vec{u}, \vec{x}^{\,i}) + \frac{1}{|\mathcal{C}_k|^2} \sum_{\vec{u} \in \mathcal{C}_k} \sum_{\vec{v} \in \mathcal{C}_k} \kappa(\vec{u}, \vec{v}).
\end{aligned}$$

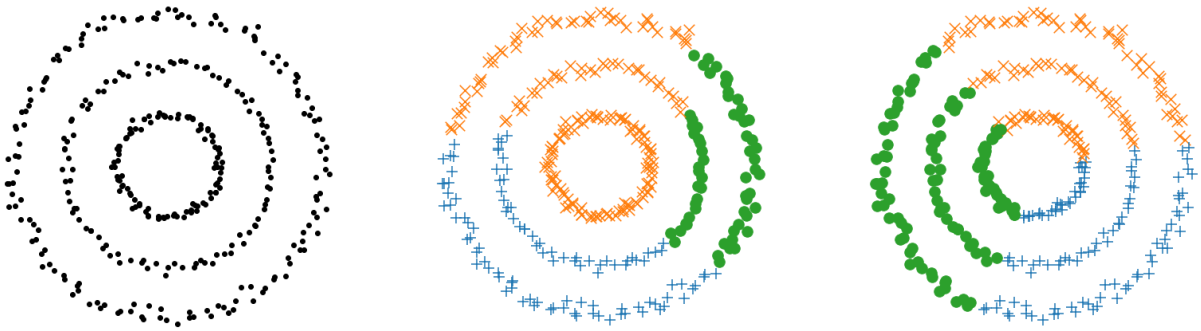
On peut ainsi réécrire l'étape (2) de l'algorithme de Lloyd sans faire appel à l'application ϕ, et sans avoir besoin de recalculer explicitement les centroïdes à l'étape (3). □

Remarque

La version à noyau de la méthode des k-moyennes ne permet généralement pas de connaître les centroïdes des clusters, puisqu'ils vivent dans l'espace de redescription $\mathcal{H}$ qui n'est pas accessible sans connaître ϕ.

12.5 Clustering par densité

Une autre façon d'obtenir des clusters non convexes est le clustering par densité. Prenons l'exemple présenté sur la figure 12.4 : il semblerait naturel de partitionner les données en 3 cercles concentriques, ce que ni le clustering agglomératif ni la méthode des k-moyennes n'arrivent à faire, car ces clusters ne sont pas convexes.



(A) Il nous semble naturel de partitionner ces données en 3 cercles concentriques.

(B) Partitionnement en 3 clusters par clustering agglomératif (lien moyen).

(C) Partitionnement en 3 clusters par k-moyennes.

FIGURE 12.4 – Motivation du clustering par densité.

C'est le problème que le *clustering par densité* cherche à résoudre, en observant que les points les plus proches d'un point donné sont sur le même cercle et non sur un autre cercle. Il s'agit de former des clusters d'observations proches les unes des autres, au sens où si deux éléments $\vec{x}^{\,i}$ et $\vec{x}^{\,l}$ d'un même cluster $\mathcal{C}_k$ peuvent être éloignés l'un de l'autre, il existe une séquence d'éléments $\vec{u}^{\,1}, \vec{u}^{\,2}, \ldots, \vec{u}^{\,m} \in \mathcal{C}_k$ tels que $\vec{u}^{\,1}$ est proche de $\vec{x}^{\,i}$, $\vec{u}^{\,m}$ est proche de $\vec{x}^{\,l}$, et pour tout $t \in \{1, \ldots, m\}$, $\vec{u}^{\,t}$ est proche de $\vec{u}^{\,t-1}$. Cette idée est illustrée sur la figure 12.5

Pour formaliser cette idée, nous allons avoir besoin de quelques définitions.

Définition 12.13 (ϵ-voisinage) Soient $\mathcal{D} = \{\vec{x}^{\,1}, \vec{x}^{\,2}, \ldots, \vec{x}^{\,n}\}$ le jeu d'éléments de $\mathcal{X}$ à partitionner, d une distance sur $\mathcal{X}$, et $\epsilon > 0$. On appelle *ϵ-voisinage* d'un élément $\vec{x} \in \mathcal{X}$ l'ensemble des observations de $\mathcal{D}$ dont la distance à $\vec{x}$ est inférieure à ϵ :

$$\mathcal{N}_\epsilon(\vec{x}) = \{\vec{u} \in \mathcal{D} : d(\vec{x}, \vec{u}) < \epsilon\}.$$

■

Définition 12.14 (Point intérieur) Étant donné $n_{\min} \in \mathbb{N}$, on dit de $\vec{x} \in \mathcal{X}$ qu'il est un *point intérieur*, ou *core point* en anglais, si son ϵ-voisinage contient au moins $n_{\min}$ éléments : $|\mathcal{N}_\epsilon(\vec{x})| \geq n_{\min}$. ■

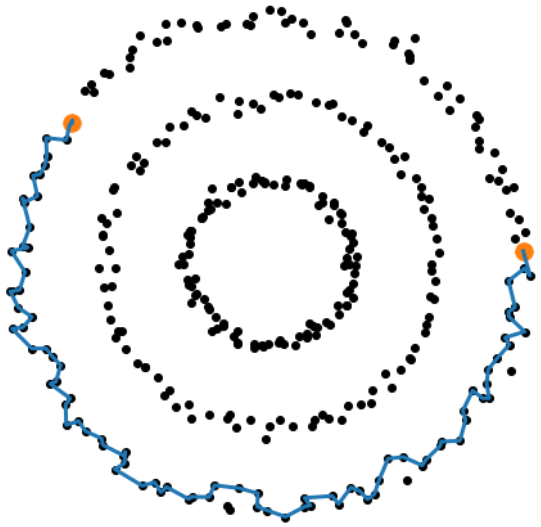


FIGURE 12.5 – Il existe un chemin entre voisins proches permettant de passer d'un point à un autre du même cluster.

Définition 12.15 (Connexion par densité) On dit que $\vec{x}$ et $\vec{v} \in \mathcal{X}$ sont *connectés par densité* si l'on peut les « relier » par une suite d'ϵ-voisinages contenant chacun au moins $n_{\min}$ éléments. Formellement, cela signifie qu'il existe $m \in \mathbb{N}$, et une séquence $\vec{u}^1, \vec{u}^2, \ldots, \vec{u}^m$ d'éléments de $\mathcal{D}$ tels que $\vec{u}^1$ est un point intérieur de $\mathcal{N}_\epsilon(\vec{x})$, $\vec{u}^2$ est un point intérieur de $\mathcal{N}_\epsilon(\vec{u}^1), \ldots,$ $\vec{u}^m$ est un point intérieur de $\mathcal{N}_\epsilon(\vec{u}^{m-1})$, $\vec{v}$ est un point intérieur de $\mathcal{N}_\epsilon(\vec{u}^m)$. ■

L'algorithme *DBSCAN*, proposé en 1996 par Martin Ester, Hans-Peter Kriegel, Jörg Sander et Xiaowei Xu (Ester et al., 1996), partitionne les données en créant des clusters de points atteignables par densité les uns depuis les autres. C'est un algorithme de partitionnement populaire, qui a obtenu en 2014 une distinction de contribution scientifique ayant résisté à l'épreuve du temps, le *test of time award* de la prestigieuse conférence KDD.

Définition 12.16 (DBSCAN) On appelle *DBSCAN*, pour *Density-Based Spatial Clustering of Applications with Noise*, ou partitionnement dans l'espace par densité pour des applications bruitées, la procédure de partitionnement suivante :

— Initialisation : Un ensemble d'éléments visités $\mathcal{V} = \emptyset$, une liste de clusters $\mathcal{K} = \emptyset$, une liste d'observations aberrantes $\mathcal{A} = \emptyset$.

— Pour chaque élément $\vec{x} \in \mathcal{D} \setminus \mathcal{V}$:

1. Construire $\mathcal{N}_\epsilon(\vec{x})$
2. Si $|\mathcal{N}_\epsilon(\vec{x})| < n_{\min}$: considérer (temporairement) $\vec{x}$ comme une observation aberrante :

$$\mathcal{A} \leftarrow \mathcal{A} \cup \{\vec{x}\}$$

   Sinon :
   — créer un cluster $\mathcal{C} \leftarrow \{\vec{x}\}$
   — augmenter ce cluster par la procédure `grow_cluster`$(\mathcal{C}, \mathcal{N}_\epsilon(\vec{x}), \epsilon, n_{\min})$
3. Ajouter $\mathcal{C}$ à la liste des clusters : $\mathcal{K} \leftarrow \mathcal{K} \cup \mathcal{C}$

4. Marquer tous les éléments de $\mathcal{C}$ comme visités : $\mathcal{V} \leftarrow \mathcal{V} \cup \mathcal{C}$.

La procédure `grow_cluster`$(\mathcal{C}, \mathcal{N}, \epsilon, n_{\min})$ est définie comme suit :
Pour tout $\vec{u} \in \mathcal{N} \setminus \mathcal{V}$:
— créer $\mathcal{N}' \leftarrow \mathcal{N}_\epsilon(\vec{u})$
— si $|\mathcal{N}'| \geq n_{\min}$: actualiser $\mathcal{N}$ de sorte à prendre les éléments de $\mathcal{N}'$ en considération : $\mathcal{N} \leftarrow \mathcal{N} \cup \mathcal{N}'$
— si $\vec{u}$ n'appartient à aucun autre cluster, l'ajouter à $\mathcal{C}$: $\mathcal{C} \leftarrow \mathcal{C} \cup \{\vec{u}\}$
— si $\vec{u}$ était précédemment cataloguée comme aberrante, l'enlever de la liste des observations aberrantes : $\mathcal{A} = \mathcal{A} \setminus \{\vec{u}\}$.

■

Un des avantages de DBSCAN est sa robustesse aux données aberrantes, qui sont identifiées lors de la formation des clusters.

Le fléau de la dimension (voir section 11.1.3) rend DBSCAN difficile à appliquer en très grande dimension : les ϵ-voisinages auront tendance à ne contenir que leur centre. De plus, la densité étant définie par les paramètres ϵ et $n_{\min}$, DBSCAN ne pourra pas trouver de clusters de densité différente. Cependant, DBSCAN ne requiert pas de prédéfinir le nombre de clusters, et est efficace en temps de calcul. Son application aux données de la figure 12.4a est illustrée sur la figure 12.6.

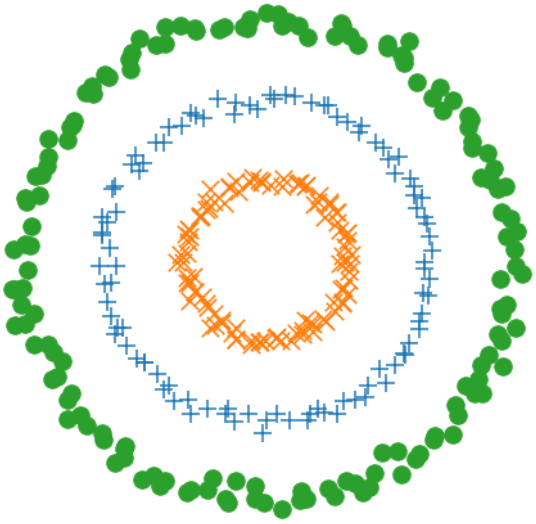


FIGURE 12.6 – Partitionnement des données de la figure 12.4a par DBSCAN.

Points clefs

— Le clustering, ou partitionnement de données, cherche à identifier des classes sans utiliser d'étiquettes.

— En l'absence d'étiquette, la qualité d'une partition peut s'évaluer sur des critères de séparabilité et d'homogénéité

— Le clustering hiérarchique partitionne les données de manière itérative. Son résultat peut être visualisé sur un dendrogramme.

— Le clustering par la méthode des k-moyennes s'effectue grâce à l'algorithme de Lloyd ou une de ses variantes. Il permet de trouver efficacement K clusters convexes.

— La version à noyau de la méthode des k-moyennes permet de l'appliquer pour découvrir des clusters non convexes.

— Le clustering par densité permet d'identifier des régions denses du jeu de données, c'est-à-dire des observations qui peuvent former un ensemble non convexe mais qui sont proches les unes des autres.

Pour aller plus loin

- Les algorithmes de partitionnement que nous avons vus associent un seul et unique cluster à chaque observation. Il existe des variantes, appelées *partitionnement flou*, ou *fuzzy clustering* en anglais, qui associent à chaque observation et chaque cluster la probabilité que cette observation appartienne à ce cluster.
- L'ouvrage de Jain et Dubes (1988) constitue une bonne introduction au clustering. On pourra aussi se reporter à l'article de Xu et Wunsch II (2005)

Bibliographie

Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231, Portland (OR). AAAI Press.

Jain, A. K. and Dubes, R. C. (1988). *Algorithms for Clustering Data*. Prentice Hall, New York.

Lloyd, S. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2) :129–137.

Steinhaus, H. (1957). Sur la division des corps matériels en parties. *Bulletin de l'Académie polonaise des Sciences*, 4(12) :801–804.

Xu, R. and Wunsch II, D. (2005). Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16 :645–678.

Appendice A

Notions d'optimisation convexe

Dans cet ouvrage, nous formulons de nombreux modèles de machine learning comme la solution d'un problème d'optimisation. Il s'agit généralement de minimiser un risque empirique, ou de maximiser une variance, souvent sous certaines contraintes. Ces problèmes sont difficiles à résoudre efficacement dans le cas général. Cependant, dans le contexte de l'apprentissage automatique, les fonctions considérées sont souvent convexes, ce qui facilite grandement la tâche. Le but de cet appendice est de présenter les problèmes d'optimisation dits convexes et quelques techniques permettant de les résoudre.

Objectifs

- Reconnaître un problème d'optimisation convexe ;
- Résoudre un problème d'optimisation convexe exactement, ou par un algorithme à directions de descente ;
- Formuler le problème dual d'un problème d'optimisation quadratique ;
- Écrire les conditions de Karush-Kuhn-Tucker pour un problème d'optimisation quadratique.

A.1 Convexité

Commençons par définir la notion de *convexité*, pour un ensemble puis pour une fonction.

A.1.1 Ensemble convexe

Définition 1.1 (Ensemble convexe) On dit d'un ensemble $\mathcal{S} \subseteq \mathbb{R}^n$ qu'il est *convexe* si et seulement si, quels que soient $\vec{u}, \vec{v} \in \mathcal{S}$ et $t \in [0, 1]$,

$$t\vec{u} + (1-t)\vec{v} \in \mathcal{S}.$$

En d'autres termes, le segment $[\vec{u}, \vec{v}]$ est entièrement contenu dans $\mathcal{S}$. ■

La figure A.1 présente quelques exemples d'ensembles convexes et non convexes dans $\mathbb{R}^2$.

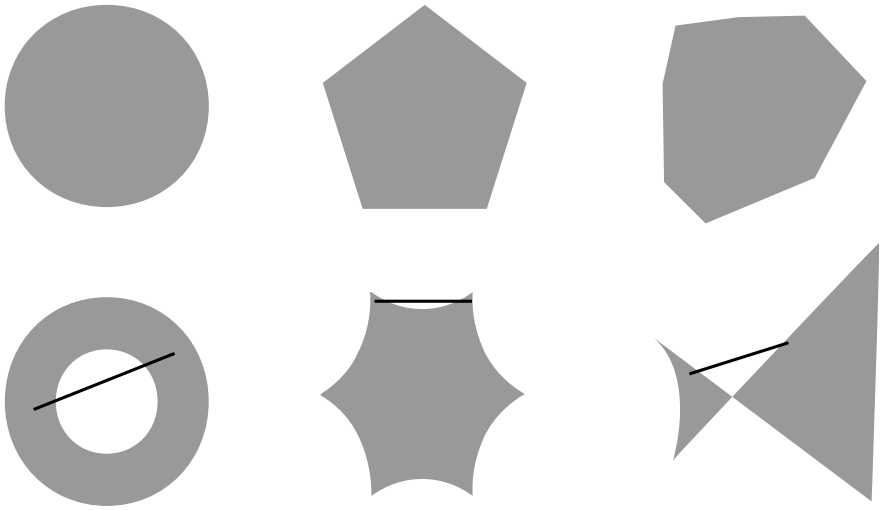


FIGURE A.1 – Les trois ensembles de $\mathbb{R}^2$ présentés sur la rangée du haut sont convexes. Les trois ensembles sur la rangée du bas ne sont pas convexes, et un exemple de segment reliant deux points de l'ensemble mais n'étant pas entièrement inclus dans cet ensemble est présenté pour chacun d'entre eux.

A.1.2 Fonction convexe

Définition 1.2 (Fonction convexe) Soit $\mathcal{U} \subseteq \mathbb{R}^n$. Une fonction $f : \mathcal{U} \to \mathbb{R}$ est dite *convexe* lorsque
— le domaine de définition $\mathcal{U}$ de f est un ensemble convexe ;
— quels que soient $\vec{u}, \vec{v} \in \mathcal{U}$ et $t \in [0, 1]$,

$$f(t\ \vec{u} + (1-t)\ \vec{v}) \leq tf(\vec{u}) + (1-t)f(\vec{v}),$$

c'est-à-dire que, sur $[\vec{u}, \vec{v}]$, f se situe au-dessous du segment $[(\vec{u}, f(\vec{u})), (\vec{v}, f(\vec{v}))]$.
Si l'inégalité est stricte pour tout $\vec{u} \neq \vec{v} in \mathcal{U}$ et $t \in]0, 1[$, on parle alors de fonction *strictement convexe*. Une fonction strictement convexe a une courbure supérieure à celle d'une fonction affine.

Enfin, dans le cas où il existe $k > 0$ tel que $f - \frac{k}{2}\,||\vec{u}||_2^2$ est strictement convexe, f est dite *fortement convexe*. ■

Exemple

Les fonctions ci-dessous sont convexes
— $f : \mathbb{R} \to \mathbb{R}, u \mapsto u^{2a} \qquad a \in \mathbb{N}$
— $f : \mathbb{R}_+^* \to \mathbb{R}, u \mapsto u^a \qquad a \notin]0, 1[$
— $f : \mathbb{R} \to \mathbb{R}, u \mapsto e^{au} \qquad a \in \mathbb{R}$
— $f : \mathbb{R}_+^* \to \mathbb{R}, u \mapsto -\log(au) \qquad a \in \mathbb{R}$
— $f : \mathbb{R}^n \to \mathbb{R}, \vec{u} \mapsto \vec{a}^{\top}\vec{u} + b \qquad \vec{a} \in \mathbb{R}^n, b \in \mathbb{R}$
— $f : \mathbb{R}^n \to \mathbb{R}, \vec{u} \mapsto \frac{1}{2}\vec{u}^{\top}Q\vec{u} + \vec{a}^{\top}\vec{u} + b \qquad \vec{a} \in \mathbb{R}^n, b \in \mathbb{R},\ Q \succeq 0$
— $f : \mathbb{R}^n \to \mathbb{R}, \vec{u} \mapsto ||\vec{u}||_p = (\sum_{i=1}^n |u_i|^p)^{\frac{1}{p}} \qquad p \geq 1.$

Une fonction dont l'opposée est convexe est dite *concave*.

Définition 1.3 (Fonction concave) Soit $\mathcal{U} \subseteq \mathbb{R}^n$. Une fonction $f : \mathcal{U} \to \mathbb{R}$ est dite *concave* si et seulement $-f$ est convexe :
— $\mathcal{U}$ est un ensemble convexe ;

— quels que soient $\vec{u}, \vec{v} \in \mathcal{U}$ et $t \in [0,1]$,

$$f(t\vec{u} + (1-t)\vec{v}) \geq tf(\vec{u}) + (1-t)f(\vec{v}).$$

Si l'inégalité est stricte pour tout $\vec{u} \neq \vec{v} in \mathcal{U}$ et $t \in]0,1[$, on parle alors de fonction *strictement concave.* ■

Les opérations suivantes préservent la convexité :

— la combinaison linéaire positive : si $f_1, f_2, \ldots, f_k : \mathbb{R}^n \to \mathbb{R}$ sont convexes, et $a_1, a_2, \ldots, a_k > 0$, alors

$$a_1 f_1 + a_2 f_2 + \cdots + a_k f_k$$

est convexe.

— la maximisation : si $f_1, f_2, \ldots, f_k : \mathbb{R}^n \to \mathbb{R}$ sont convexes, alors

$$\vec{u} \mapsto \max_{1,\ldots,k} f_k(\vec{u})$$

est convexe.

— la minimisation partielle : si $f : \mathbb{R}^n \to \mathbb{R}$ est convexe, et $\mathcal{C} \subseteq \mathbb{R}^n$ est un ensemble convexe, alors

$$u_1, u_2, \ldots, u_{k-1} \mapsto \min_{v \in \mathcal{C}} f(u_1, u_2, \ldots, u_{k-1}, v)$$

est convexe.

A.2 Problèmes d'optimisation convexe

A.2.1 Formulation et vocabulaire

Définition 1.4 (Optimisation convexe) Étant donnés $\mathcal{U} \subseteq \mathbb{R}^n$ et une fonction $f : \mathcal{U} \to \mathbb{R}$ convexe, on appelle *problème d'optimisation convexe* le problème suivant

$$\min_{\vec{u} \in \mathcal{U}} f(\vec{u}).$$

On appelle alors f la *fonction objectif*, ou encore la *fonction coût* ou la *fonction critère.*

Un point $\vec{u}^* \in \mathcal{U}$ vérifiant $f(\vec{u}^*) \leq f(\vec{u})\ \forall \vec{u} \in \mathcal{U}$ est appelé *point minimum* de f sur $\mathcal{U}$, tandis que la valeur $f(\vec{u}^*)$ est appelée *minimum* de f sur $\mathcal{U}$. ■

Définition 1.5 (Optimisation convexe sous contraintes) Étant donnés m, r deux entiers positifs, des ensembles $\mathcal{U}, \mathcal{U}_1, \ldots, \mathcal{U}_m, \mathcal{V}_1, \ldots, \mathcal{V}_r \subseteq \mathbb{R}^n$, une fonction convexe $f : \mathcal{U} \to \mathbb{R}$, m fonctions $g_i : \mathcal{U}_i \to \mathbb{R}$, convexes aussi, et r fonctions $h_j : \mathcal{V}_j \to \mathbb{R}$ affines, on appelle *problème d'optimisation convexe sous contraintes* le problème suivant

$$\begin{aligned} &\min_{\vec{u} \in \mathcal{D}} f(\vec{u}) \text{ sous les contraintes} \\ &\quad g_i(\vec{u}) \leq 0 \quad \forall i = 1, \ldots, m \\ &\quad h_j(\vec{u}) = 0 \quad \forall j = 1, \ldots, r, \end{aligned}$$

où $\mathcal{D} = \mathcal{U} \cap \bigcap_{i=1}^m \mathcal{U}_i \bigcap_{j=1}^r \mathcal{V}_j$ est le domaine commun à toutes les fonctions considérées.

f est la *fonction objectif.* Les m contraintes $g_i \leq 0$ sont les *contraintes d'inégalité.* Les r contraintes $h_j = 0$ sont les *contraintes d'égalité.*

$\vec{v} \in \mathcal{D}$ qui vérifie toutes les contraintes est appelé un *point admissible* ou *point réalisable* (*feasible point* en anglais), et l'ensemble des points admissibles est appelé *la région admissible* ou *le domaine des contraintes* (*feasible domain* en anglais). Cet ensemble est convexe.

■

A.2.2 Extrema locaux et globaux

Le terme d'extremum désigne indifféremment un maximum ou un minimum. Ceux-ci peuvent être globaux, c'est-à-dire maximiser (respectivement minimiser) la fonction sur tout son domaine, ou locaux, c'est à dire optimaux dans leur voisinage. Formellement :

Définition 1.6 (Extremum global) Soit $\mathcal{U} \subseteq \mathbb{R}^n$, $f : \mathcal{U} \to \mathbb{R}$, et $\vec{u}^* \in \mathcal{U}$. On dit que $\vec{u}^*$ est un *point de minimum global*, ou *point de minimum absolu* de f sur $\mathcal{U}$ si

$$f(\vec{u}^*) \leq f(\vec{u}) \quad \forall \vec{u} \in \mathcal{U}.$$

$\vec{u}^*$ est un *point de maximum global* de f sur $\mathcal{U}$ s'il est un point de minimum global de $-f$ sur $\mathcal{U}$. ■

Définition 1.7 (Extremum local) Soient $\mathcal{U} \subseteq \mathbb{R}^n$, $f : \mathcal{U} \to \mathbb{R}$, et $\vec{u}^* \in \mathcal{U}$. On dit que $\vec{u}^*$ est un *point de minimum local*, ou *point de minimum relatif* de f sur $\mathcal{U}$ s'il existe un voisinage $\mathcal{V}$ de $\vec{u}^*$ dans $\mathbb{R}^n$ tel que

$$f(\vec{u}^*) \leq f(\vec{u}) \quad \forall \vec{u} \in \mathcal{U} \cap \mathcal{V}.$$

$\vec{u}^*$ est un *point de maximum local* de f sur $\mathcal{U}$ s'il est un point de minimum local de $-f$ sur $\mathcal{U}$. ■

La notion de convexité est très importante en optimisation, car elle permet de garantir que les minima locaux sont des minima globaux.

Théorème 1.1 *Soit $\mathcal{U} \subseteq \mathbb{R}^n$, $f : \mathcal{U} \to \mathbb{R}$, et $\vec{u}^* \in \mathcal{U}$ un point de minimum local de f sur $\mathcal{U}$. Alors*
- *Si f est convexe, alors $\vec{u}^*$ est un point de minimum global ;*
- *Si f est strictement convexe, alors $\vec{u}^*$ est l'unique point de minimum global.*

■

Démonstration. $\vec{u}^*$ étant un point de minimum local, il existe $\epsilon > 0$ tel que

$$\text{pour tout } \vec{u} \text{ vérifiant } ||\vec{u} - \vec{u}^*||_2 \leq \epsilon, f(\vec{u}^*) \leq f(\vec{u}). \tag{A.1}$$

Soit $\vec{v} \in \mathcal{U}$ différent de $\vec{u}^*$. Supposons f convexe. Nous allons montrer que $f(\vec{v}) \geq f(\vec{u}^*)$. Deux cas sont possibles : soit $\vec{v}$ appartient au voisinage de $\vec{u}^*$, autrement dit $||\vec{v} - \vec{u}^*||_2 \leq \epsilon$, auquel cas $f(\vec{v}) \geq f(\vec{u}^*)$ d'après l'équation A.1, soit $\vec{v}$ est à l'extérieur de ce voisinage, auquel cas

$$||\vec{v} - \vec{u}^*||_2 > \epsilon. \tag{A.2}$$

Définissons maintenant $\vec{u} = (1-\lambda)\vec{u}^* + \lambda\vec{v}$, avec $\lambda = \frac{1}{2}\frac{\epsilon}{||\vec{v}-\vec{u}^*||_2}$. D'après l'équation A.2, $0 < \lambda < 1$. Par convexité de $\mathcal{U}$, $\vec{u} \in \mathcal{U}$.

$||\vec{u} - \vec{u}^*||_2 = ||(1-\lambda)\vec{u}^* + \lambda\vec{v} - \vec{u}^*||_2 = \lambda\,||\vec{v} - \vec{u}^*||_2 = \frac{\epsilon}{2} < \epsilon$, donc $\vec{u}$ appartient au voisinage de $\vec{u}^*$. D'après l'équation A.1, $f(\vec{u}) \geq f(\vec{u}^*)$.

Par convexité de f, $f(\vec{u}) \leq (1-\lambda)f(\vec{u}^*) + \lambda f(\vec{v}) = f(\vec{u}^*) + \lambda(f(\vec{v}) - f(\vec{u}^*))$. Comme $f(\vec{u}^*) \leq f(\vec{u})$, on obtient $f(\vec{u}^*) \leq f(\vec{u}^*) + \lambda(f(\vec{v}) - f(\vec{u}^*))$, ce dont on déduit (comme $\lambda > 0$) que $f(\vec{v}) \geq f(\vec{u}^*)$.

Ainsi, pour tout $\vec{v} \in \mathcal{U}$ différent de $\vec{u}^*$, $f(\vec{v}) \geq f(\vec{u}^*)$ et $\vec{u}^*$ est donc bien un minimum global de f. Dans le cas où f est strictement convexe, le même raisonnement tient avec des inégalités strictes et on obtient alors que $\vec{u}^*$ est un unique minimum global de f. □

Nous allons maintenant voir comment résoudre un problème d'optimisation convexe.

A.3 Optimisation convexe sans contrainte

Commençons par les problèmes d'optimisation convexe sans contrainte. Bien que l'optimisation convexe soit difficile dans le cas général, il existe de nombreuses techniques permettant d'obtenir efficacement une solution numérique approchée de très bonne qualité, notamment sous certaines hypothèses de continuité et de différentiabilité souvent vérifiées pour les problèmes posés dans le cadre de l'apprentissage statistique.

A.3.1 Caractérisation différentielle de la convexité

Une fonction convexe et différentiable se situe au-dessus de n'importe laquelle de ses tangentes, comme illustré sur la figure A.2. Formellement :

Théorème 1.2 Caractérisation du premier ordre *Soit $\mathcal{U} \subseteq \mathbb{R}^n$ et une fonction $f : \mathcal{U} \to \mathbb{R}$ de classe C^1 (autrement dit continue, et dont toutes les dérivées partielles existent et sont continues). f est convexe si et seulement si*

- *$\mathcal{U}$ est convexe ;*
- *quels que soient $\vec{u}, \vec{v} \in \mathcal{U}$,*

$$f(\vec{v}) \geq f(\vec{u}) + \nabla f(\vec{u})^\top (\vec{v} - \vec{u}). \tag{A.3}$$

■

Démonstration. Soient $t \in [0, 1]$ et $\vec{u}, \vec{v} \in \mathcal{U}$. Supposons f convexe. Alors

$$\lim_{t \to 0^+} \frac{1}{t} \left(f(\vec{u} + t(\vec{v} - \vec{u})) - f(\vec{u}) \right) = \nabla f(\vec{u})^\top (\vec{v} - \vec{u}). \tag{A.4}$$

Par convexité de f, on peut écrire

$$f(\vec{u} + t(\vec{v} - \vec{u})) = f((1 - t)\vec{u} + t\vec{v}) \leq (1 - t) f(\vec{u}) + t f(\vec{v}), \tag{A.5}$$

et donc

$$\frac{1}{t} \left(f(\vec{u} + t(\vec{v} - \vec{u})) - f(\vec{u}) \right) \leq f(\vec{v}) - f(\vec{u}). \tag{A.6}$$

En passant à la limite dans l'équation A.4 on obtient l'inégalité A.3.

Réciproquement, étant donnés $\vec{u}, \vec{v} \in \mathcal{U}$ et $t \in]0, 1[$, posons $\vec{w} = t\vec{u} + (1 - t)\vec{v}$. Par convexité de $\mathcal{U}$, $\vec{w} \in \mathcal{U}$. En supposant l'inégalité A.3,

$$f(\vec{u}) \geq f(\vec{w}) + \nabla f(\vec{w})^\top (\vec{u} - \vec{w})$$
$$f(\vec{v}) \geq f(\vec{w}) + \nabla f(\vec{w})^\top (\vec{v} - \vec{w}).$$

En additionnant la première inégalité multipliée par t à la deuxième multipliée par $(1 - t)$, on obtient

$$t f(\vec{u}) + (1 - t) f(\vec{v}) \leq f(\vec{w}) + \nabla f(\vec{w})^\top (\vec{w} - \vec{w}) \tag{A.7}$$

et donc la convexité de f. □

Il découle du théorème 1.2 qu'*une fonction f convexe et différentiable est minimale là où son gradient s'annule.* Formellement :

Théorème 1.3 *Soit $\mathcal{U} \subseteq \mathbb{R}^n$ et une fonction $f : \mathcal{U} \to \mathbb{R}$ convexe, de classe C^1. Soit $\vec{u}^* \in \mathcal{U}$. Les propositions suivantes sont équivalentes :*

1. *$\vec{u}^*$ est un point de minimum de f sur $\mathcal{U}$;*
2. *$\nabla f(\vec{u}^*) = 0$.*

■

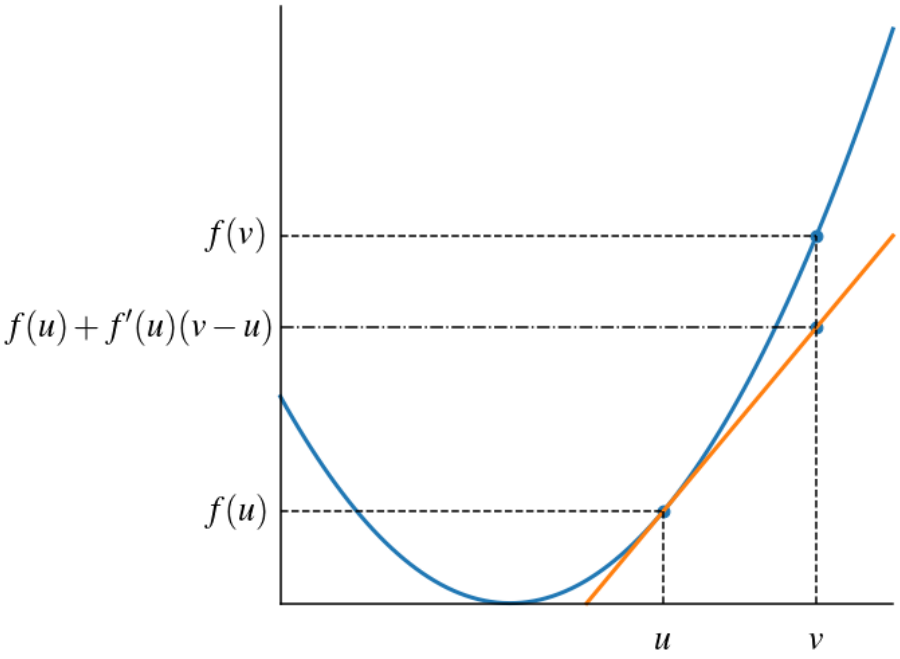


FIGURE A.2 – La fonction convexe f est située au-dessus de sa tangente en u.

A.3.2 Caractérisation du deuxième ordre de la convexité

Les fonctions convexes de classe $\mathcal{C}^2$ sont caractérisées par le fait que leur hessienne est semi-définie positive. En d'autres termes, elles sont de courbure positive partout. Cette propriété est exploitée par certain des algorithmes présentés plus bas.

Théorème 1.4 Caractérisation du deuxième ordre *Soit $\mathcal{U} \subseteq \mathbb{R}^n$ et $f : \mathcal{U} \to \mathbb{R}$ une fonction de classe $\mathcal{C}^2$ (autrement dit continue, deux fois dérivable et de différentielles d'ordre 1 et 2 continues). f est convexe si et seulement si*

- *$\mathcal{U}$ est convexe ;*
- *quel que soit $\vec{u} \in \mathcal{U}$, $\nabla^2 f(\vec{u}) \succeq 0$.*

■

Démonstration. Soient $\vec{u} \in \mathcal{U}$ et $\vec{v} \in \mathbb{R}^n$. Posons $I = \{t \in \mathbb{R} | \vec{u} + t\vec{v} \in \mathcal{U}\}$ et définissons $\phi : I \to \mathbb{R}, t \mapsto f(\vec{u} + t\vec{v})$. Alors I est un intervalle et la fonction ϕ est convexe.

En effet, supposons que I contient au moins deux points distincts t_1 et t_2. (Dans le cas contraire, I est soit un singleton, soit l'intervalle vide. C'est donc un intervalle, et de plus, la convexité de ϕ est triviale.) Soit $s \in]0,1[$ et $\vec{w} = \vec{u} + (st_1 + (1-s)t_2)\,\vec{v}$. Alors $\vec{w} = s\vec{u}_1 + (1-s)\vec{u}_2$ avec $\vec{u}_1 = \vec{u} + t_1\vec{v}$ et $\vec{u}_2 = \vec{u} + t_2\vec{v}$. Comme $t_1 \neq t_2$, $\vec{u}_1 \neq \vec{u}_2$ et par convexité de $\mathcal{U}$, $\vec{w} \in \mathcal{U}$ donc $(st_1 + (1-s)t_2) \in I$. Ce raisonnement étant vrai pour tout $s \in]0,1[$, I est donc un intervalle. De plus, par convexité de f,

$$\begin{aligned}\phi\left(st_1 + (1-s)t_2\right) = f\left(s\vec{u}_1 + (1-s)\vec{u}_2\right) &\leq sf(\vec{u}_1) + (1-s)f(\vec{u}_2) \\ &= s\phi(t_1) + (1-s)\phi(t_2)\end{aligned}$$

et donc ϕ est convexe.

Comme f est $\mathcal{C}^2$ sur $\mathcal{U}$, ϕ est $\mathcal{C}^2$ sur I. On a

$$\phi''(t) = \vec{v}^{\top} \nabla^2 f(\vec{u} + t\vec{v})\ \vec{v} \quad \forall t \in I. \tag{A.8}$$

Par convexité de ϕ et le théorème 1.2, ϕ' est croissante et donc $\phi'' \geq 0$. En particulier, $\phi''(0) \geq 0$ et donc $\vec{v}^{\top} \nabla^2 f(\vec{u})\ \vec{v} \geq 0$. Comme le raisonnement tient pour tout $\vec{u} \in \mathcal{U}$ et tout $\vec{v} \in \mathbb{R}^n$, $\nabla^2 f(\vec{u}) \succeq 0$.

Réciproquement, supposons $\nabla^2 f(\vec{u})$ semi-définie positive en tout point $\vec{u} \in \mathcal{U}$. Prenons $\vec{u}_1, \vec{u}_2 \in \mathcal{U}$, posons $\vec{u} = \vec{u}_2$ et $\vec{v} = \vec{u}_1 - \vec{u}_2$ et définissons $\phi : t \mapsto f(\vec{u} + t\vec{v})$ sur $I = \{t \in \mathbb{R} | \vec{u} + t\vec{v} \in \mathcal{U}\}$. Alors, par l'équation A.8, ϕ'' est positive sur I et donc ϕ est convexe sur I. Enfin, I contient les points $t_1 = 1$ et $t_2 = 0$. Alors, pour tout $s \in]0, 1[$, $s = s\ t_1 + (1 - s)t_2$ et

$$f\left(s\vec{u}_1 + (1-s)\vec{u}_2\right) = \phi(s) \leq s\phi(t_1) + (1-s)\phi(t_2) = sf(\vec{u}_1) + (1-s)f(\vec{u}_2) \tag{A.9}$$

et donc f est convexe. □

A.3.3 Algorithme du gradient

Dans le cas où la fonction convexe que l'on cherche à minimiser est différentiable, il n'est pas toujours évident de trouver un point où son gradient s'annule. Il est alors possible de trouver une solution approchée en remarquant que le gradient ∇f indique la direction de plus grande augmentation de la fonction f. Ainsi, si l'on choisit un point $\vec{u}$ au hasard dans le domaine de f, et que $\nabla f(\vec{u}) \neq 0$, un réel positif t suffisamment petit, $(\vec{u} - t\nabla f(\vec{u}))$ est plus proche du point de minimum u^* que u. Ce concept est illustré sur la figure A.3.

On dérive de cette observation toute une famille d'algorithmes, appelés *algorithmes à directions de descente,* dont la suite de cette section présente quelques exemples parmi les plus fréquemment utilisés. Le plus simple de ces algorithmes est l'*algorithme du gradient*, parfois appelé *descente de gradient* par analogie avec son nom anglais *gradient descent*.

Définition 1.8 (Algorithme du gradient) Soient $\mathcal{U} \subseteq \mathbb{R}^n$ et $f : \mathcal{U} \to \mathbb{R}$ une fonction de classe $\mathcal{C}^1$. Étant donnés un *pas* $\alpha > 0$ et une *tolérance* $\epsilon > 0$, on appelle *algorithme du gradient* l'algorithme suivant :

1. Choisir $\vec{u}$ aléatoirement dans $\mathcal{U}$.
2. Tant que $||\nabla f(\vec{u})||_2^2 > \epsilon$, actualiser $\vec{u}$:

$$\vec{u} \leftarrow \vec{u} - \alpha \nabla f(\vec{u}). \tag{A.10}$$

$\vec{u}$ est alors une approximation du point de minimum global de f sur $\mathcal{U}$. ■

Plus la tolérance est faible, plus le point de minimum sera proche numériquement du point de minimum global.

Attention

Le pas α est un paramètre très important de l'algorithme du gradient. Si α est très faible, l'algorithme mettra très longtemps à converger. À l'inverse, si α est très élevé, $\vec{u}$ oscillera autour du minimum global et l'algorithme peut même diverger.

On utilisera donc souvent un *pas adaptatif* qui varie à chaque itération et commencera par prendre des valeurs relativement élevées avant de diminuer progressivement lorsqu'on se rapproche de la solution.

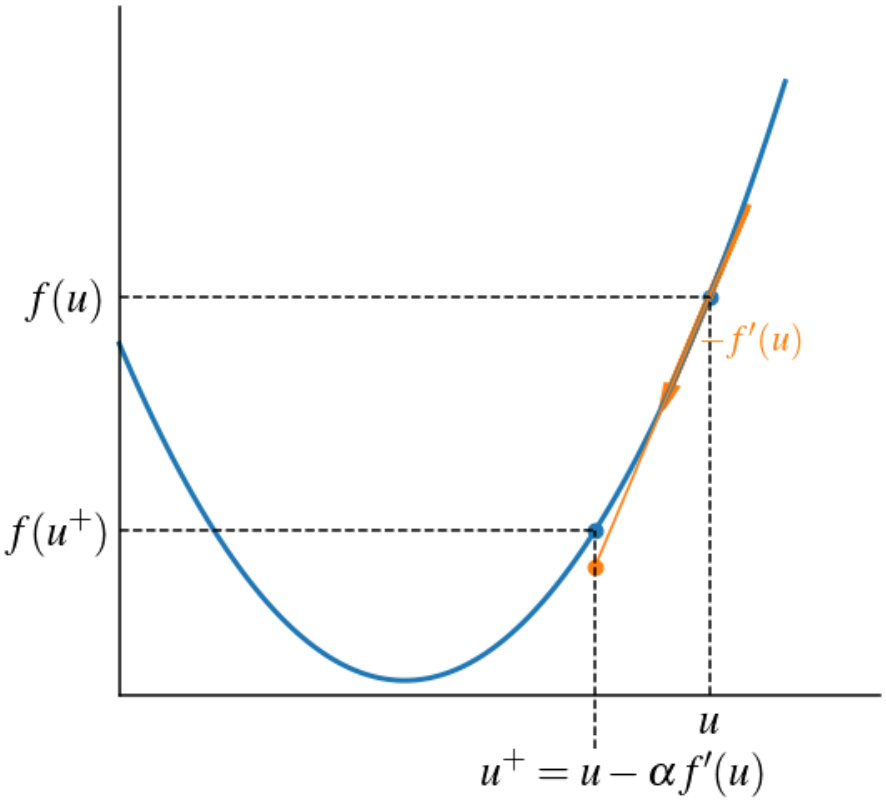


FIGURE A.3 – Une itération de l'algorithme du gradient déplace u de $-\alpha f'(u)$.

A.3.4 Recherche linéaire par rebroussement

Une méthode couramment utilisée pour adapter la taille du pas de l'algorithme du gradient est la *recherche linéaire par rebroussement*. Cette méthode repose sur l'observation, illustrée sur la figure A.4, que lorsque

$$f(\vec{u} - \alpha \nabla f(\vec{u})) \leq f(\vec{u}) - \frac{\alpha}{2} \nabla f(\vec{u})^\top \nabla f(\vec{u}),$$

il est vraisemblable que α soit suffisamment faible pour que $\vec{u}$ reste du même côté du minimum global après une itération.

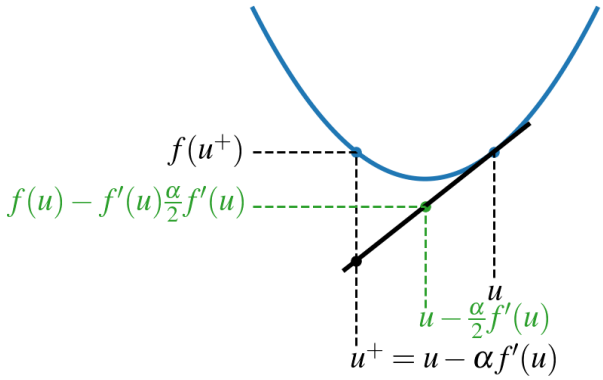
Définition 1.9 (Recherche linéaire par rebroussement) Soit $\mathcal{U} \subseteq \mathbb{R}^n$ et une fonction $f : \mathcal{U} \to \mathbb{R}$ de classe C^1. Étant donnés un *pas initial* $\alpha > 0$, un *coefficient de réduction* $\beta \in]0, 1[$ et une *tolérance* $\epsilon > 0$, on appelle *recherche linéaire par rebroussement,* ou *BLS* pour *backtracking line search* en anglais, l'algorithme suivant :

1. Choisir $\vec{u}$ aléatoirement dans $\mathcal{U}$.
2. Tant que $||\nabla f(\vec{u})||_2^2 > \epsilon$:
   — Si $f(\vec{u} - \alpha \nabla f(\vec{u})) > f(\vec{u}) - \frac{\alpha}{2} \nabla f(\vec{u})^\top \nabla f(\vec{u})$, réduire le pas :

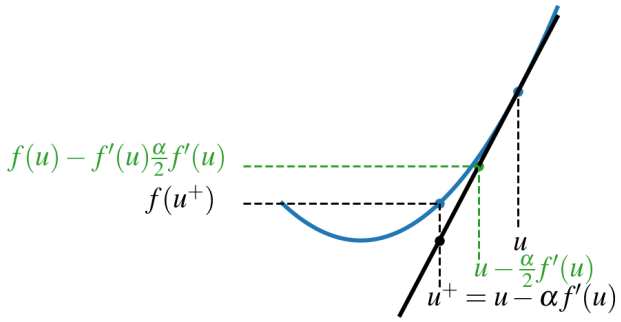
   $$\alpha \leftarrow \beta \alpha$$

   — Actualiser $\vec{u}$: $\vec{u} \leftarrow \vec{u} - \alpha \nabla f(\vec{u})$.

$\vec{u}$ est alors une approximation du point de minimum global de f sur $\mathcal{U}$. ■



(A) Quand $f(u - \alpha f'(u)) > f(u) - f'(u)\frac{\alpha}{2}f'(u)$, le pas α est trop élevé et $u - \alpha f'(u)$ va se retrouver de l'autre côté du point de minimum. Il faut donc le réduire.



(B) Quand $f(u - \alpha f'(u)) \leq f(u) - f'(u)\frac{\alpha}{2}f'(u)$, le pas α est suffisamment petit pour que $u - \alpha f'(u)$ soit entre le point de minimum et u.

FIGURE A.4 – Comparer $f(u - \alpha f'(u))$ à $f(u) - f'(u)\frac{\alpha}{2}f'(u)$ permet de déterminer si la valeur de α est trop élevée.

A.3.5 Méthode de Newton

Quand la fonction convexe à minimiser, f, est de classe $\mathcal{C}^2$, il existe des façons plus efficaces d'adapter le pas. La méthode de Newton est l'une d'entre elles et repose sur le développement de Taylor au deuxième ordre de f en $\vec{u}$. Étant donné $\vec{u} \in \mathcal{U}$, nous définissons la fonction $g : \mathcal{U} \to \mathbb{R}$ donnée par

$$g(\vec{v}) = f(\vec{u}) + \nabla f(\vec{u})^\top (\vec{v} - \vec{u}) + \frac{1}{2}(\vec{v} - \vec{u})^\top \nabla^2 f(\vec{u})(\vec{v} - \vec{u}) \tag{A.11}$$

Pour minimiser g, il suffit d'annuler son gradient

$$\nabla g(\vec{v}) = \nabla f(\vec{u}) + \nabla^2 f(\vec{u})(\vec{v} - \vec{u}) \tag{A.12}$$

et ainsi g est minimale au point

$$\vec{v}^* = \vec{u} - \left(\nabla^2 f(\vec{u})\right)^{-1} \nabla f(\vec{u}). \tag{A.13}$$

Ainsi, la méthode de Newton consiste à utiliser comme pas $\alpha = \left(\nabla^2 f(\vec{u})\right)^{-1}$. Cette méthode suppose que la hessienne est inversible, ce qui est, entre autres, le cas pour les fonctions fortement convexes. Dans le cas contraire, on pourra ajouter un peu de bruit à la hessienne (en lui ajoutant ϵI_n, avec $\epsilon > 0$ petit) afin de la rendre inversible.

Définition 1.10 (Méthode de Newton) Soient $\mathcal{U} \subseteq \mathbb{R}^n$ et $f : \mathcal{U} \to \mathbb{R}$ une fonction de classe $\mathcal{C}^1$. Étant donnée une *tolérance* $\epsilon > 0$, on appelle *méthode de Newton* l'algorithme suivant :

1. Choisir $\vec{u}$ aléatoirement dans $\mathcal{U}$.
2. Tant que $||\nabla f(\vec{u})||_2^2 > \epsilon$:
   - Calculer le pas : $\alpha = \left(\nabla^2 f(\vec{u})\right)^{-1}$.
   - Actualiser $\vec{u}$: $\vec{u} \leftarrow \vec{u} - \alpha \nabla f(\vec{u})$.

$\vec{u}$ est alors une approximation du point de minimum global de f sur $\mathcal{U}$. ■

A.3.6 Méthode de Newton par gradient conjugué

La méthode de Newton peut être lourde à mettre en œuvre en raison du calcul de l'inverse de la hessienne. Pour y remédier, la *méthode de Newton par gradient conjugué* consiste à calculer à chaque étape la quantité δ solution de

$$\nabla^2 f(\vec{u})\delta = \nabla f(\vec{u}). \tag{A.14}$$

La règle d'actualisation de $\vec{u}$ dans l'algorithme du gradient devient alors $\vec{u} \leftarrow \vec{u} - \delta$.

L'équation A.14 est un problème de la forme $A\vec{x} - \vec{b} = 0$, où $A \succeq 0$ (d'après le théorème 1.4). C'est cette équation que l'on résout en utilisant la méthode dite du *gradient conjugué*. Cette méthode a été proposée dans les années 1950 par Cornelius Lanczos, Eduard Stiefel et Magnus Hestenes (Hestenes et Stiefel, 1952).

L'idée centrale de cette méthode est de construire une base de $\mathbb{R}^n$ constituée de vecteurs conjugués par rapport à A :

$$\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n\} \text{ tels que } \vec{v}_i^\top A \vec{v}_j = 0 \quad \forall i \neq j. \tag{A.15}$$

Définition 1.11 (Méthode du gradient conjugué) Étant donnés une matrice semi-définie positive $A \in \mathbb{R}^{n\times n}$ et un vecteur $b \in \mathbb{R}^n$, la *méthode du gradient conjugué*, ou *conjugate gradient* en anglais, est l'algorithme suivant :

1. Initialisation :
   - Choisir aléatoirement $\vec{x}^{\,(0)} \in \mathbb{R}^n$.
   - Initialiser $\vec{r}_0 = \vec{v}_0 = \vec{b} - A\vec{x}^{\,(0)}$.
2. Pour $t = 1, \ldots, n$:
   (a) Actualiser $\vec{x}$:
   $$\vec{x}^{\,(t)} = \vec{x}^{\,(t-1)} + \frac{\vec{r}_{t-1}^\top \vec{r}_{t-1}}{\vec{v}_{t-1}^\top A \vec{v}_{t-1}} \vec{v}_{t-1}. \tag{A.16}$$
   (b) Actualiser le résiduel :
   $$\vec{r}_t = \vec{b} - A\vec{x}^{\,(t)}. \tag{A.17}$$
   (c) Actualiser $\vec{v}$:
   $$\vec{v}_t = \vec{r}_t + \frac{\vec{r}_t^\top \vec{r}_t}{\vec{r}_{t-1}^\top \vec{r}_{t-1}} \vec{v}_{t-1}. \tag{A.18}$$

$\vec{x}^{\,(n)}$ est la solution recherchée. ■

Théorème 1.5 *La méthode du gradient conjugué assure que* $\vec{v}_i^\top A \vec{v}_j = 0 \quad \forall i \neq j$. ■

Démonstration. Posons $\alpha_t = \frac{\vec{r}_{t-1}^\top \vec{r}_{t-1}}{\vec{v}_{t-1}^\top A \vec{v}_{t-1}}$ et $\beta_t = \frac{\vec{r}_t^\top \vec{r}_t}{\vec{r}_{t-1}^\top \vec{r}_{t-1}}$.

Nous allons commencer par poser quelques réécritures des définitions ci-dessus qui nous serviront par la suite. En remplaçant $\vec{x}^{\,(t)}$ par sa définition (équation A.16) dans l'équation A.17, on obtient $\vec{r}_t = \vec{b} - A\vec{x}^{\,(t-1)} - \alpha_t A\vec{v}_{t-1}$ et donc :

$$\vec{r}_t = \vec{r}_{t-1} - \alpha_t A \vec{v}_{t-1}. \tag{A.19}$$

De manière équivalente,

$$A\vec{v}_{t-1} = \frac{1}{\alpha_t}(\vec{r}_{t-1} - \vec{r}_t). \tag{A.20}$$

De plus, d'après l'équation A.18,

$$\vec{r}_t = \vec{v}_t - \beta_t \vec{v}_{t-1}. \tag{A.21}$$

Enfin, d'après les définitions de α_t et β_t,

$$\beta_t \vec{v}_{t-1}^\top A\vec{v}_{t-1} = \frac{\vec{r}_t^\top \vec{r}_t}{\alpha_t}. \tag{A.22}$$

Nous allons maintenant montrer par récurrence que pour tout $t = 1, \dots, n$, pour tout $i = 1, \dots, t$, $\vec{r}_t^\top \vec{r}_{t-i} = 0$ et $\vec{v}_t^\top A\vec{v}_{t-i} = 0$.

Commençons par le cas $t = 1$ (et donc $i = 1$.) Il s'agit de montrer que $\vec{r}_1^\top \vec{r}_0 = 0$ et $\vec{v}_1^\top A\vec{v}_0 = 0$. D'après l'équation A.19,

$$\begin{aligned}\vec{r}_1^\top \vec{r}_0 &= \vec{r}_0^\top \vec{r}_0 - \alpha_1 A\vec{v}_0^\top \vec{r}_0 \\ &= \vec{r}_0^\top \vec{r}_0 \left(1 - \frac{\vec{v}_0^\top \vec{r}_0}{\vec{v}_0^\top \vec{v}_0} A\right).\end{aligned}$$

Comme $\vec{r}_0 = \vec{v}_0$, la fraction ci-dessus vaut 1, et on obtient bien $\vec{r}_1^\top \vec{r}_0 = 0$.

Par la définition de $\vec{v}_1$ (équation A.18), $\vec{v}_1^\top A\vec{v}_0 = \vec{r}_1^\top A\vec{v}_0 + \beta_1 \vec{v}_0^\top A\vec{v}_0$. D'après l'équation A.20, $\vec{r}_1^\top A\vec{v}_0 = \frac{1}{\alpha_1}\vec{r}_1^\top(\vec{r}_0 - \vec{r}_1)$. D'après l'équation A.22, $\beta_1 \vec{v}_0^\top A\vec{v}_0 = \frac{\vec{r}_1^\top \vec{r}_1}{\alpha_1}$. Ainsi, $\vec{v}_1^\top A\vec{v}_0 = \frac{\vec{r}_1^\top \vec{r}_0}{\alpha_1} = 0$ car nous venons de montrer que $\vec{r}_1^\top \vec{r}_0 = 0$. Ceci conclut le cas $t = 1$.

Supposons maintenant $t > 1$, et que pour tout $u < t$, pour tout $i = 1, \dots, u$, $\vec{r}_u^\top \vec{r}_{u-i} = 0$ et $\vec{v}_u^\top A\vec{v}_{u-i} = 0$.

Nous allons tout d'abord montrer que $\vec{r}_t^\top \vec{r}_{t-i} = 0$ pour tout $i = 1, \dots, t$. En utilisant l'équation A.19, et le fait que $A = A^\top$ (car $A \succeq 0$,) on obtient

$$\vec{r}_t^\top \vec{r}_{t-i} = \vec{r}_{t-1}^\top \vec{r}_{t-i} - \alpha_t \vec{v}_{t-1}^\top A\vec{r}_{t-i}. \tag{A.23}$$

Trois cas sont possibles :

— Si $i = 1$, nous avons alors, en remplaçant i par 1 et α_t par sa définition dans l'équation A.23

$$\vec{r}_t^\top \vec{r}_{t-1} = \vec{r}_{t-1}^\top \vec{r}_{t-1} \left(1 - \frac{\vec{v}_{t-1}^\top A\vec{r}_{t-1}}{\vec{v}_{t-1}^\top A\vec{v}_{t-1}}\right).$$

D'après l'équation A.21,

$$\vec{v}_{t-1}^\top A\vec{r}_{t-1} = \vec{v}_{t-1}^\top A\vec{v}_{t-1} - \beta_{t-1}\vec{v}_{t-1}^\top A\vec{v}_{t-2}.$$

D'après les hypothèses de la récurrence, $\vec{v}_{t-1}^\top A\vec{v}_{t-2} = 0$ et donc $\vec{v}_{t-1}^\top A\vec{r}_{t-1} = \vec{v}_{t-1}^\top A\vec{v}_{t-1}$, et ainsi la fraction ci-dessus est égale à 1, ce qui nous permet de conclure que $\vec{r}_t^\top \vec{r}_{t-1} = 0$.

— Si $1 < i < t$, nous pouvons remplacer la deuxième occurrence de $\vec{r}_{t-i}$ dans l'équation A.23 par sa valeur donnée par l'équation A.21, et obtenir ainsi

$$\vec{r}_t^\top \vec{r}_{t-i} = \vec{r}_{t-1}^\top \vec{r}_{t-i} - \alpha_t \vec{v}_{t-1}^\top A\vec{v}_{t-i} - \alpha_t \beta_{t-i} \vec{v}_{t-1}^\top A\vec{v}_{t-i-1}.$$

Chacun des termes de cette somme est nul d'après nos hypothèses de récurrence, ce qui nous permet de conclure que $\vec{r}_{t-1}^\top \vec{r}_{t-i} = 0$.

— Enfin, si $i = t$, il s'agit d'étudier $\vec{r}_{t-1}^\top \vec{r}_0 - \alpha_t \vec{v}_{t-1}^\top A\vec{r}_0$. Comme $\vec{r}_0 = \vec{v}_0$, les deux termes de cette somme sont nuls d'après nos hypothèses de récurrence, et $\vec{r}_t^\top \vec{r}_0 = 0$.

Nous pouvons maintenant nous intéresser à $\vec{v}_t^\top A\vec{v}_{t-i}$. D'après l'équation A.18,

$$\vec{v}_t^\top A\vec{v}_{t-i} = \vec{r}_t^\top A\vec{v}_{t-i} + \beta_t \vec{v}_{t-1}^\top A\vec{v}_{t-i}. \tag{A.24}$$

D'après l'équation A.20, le premier terme de cette somme peut s'écrire

$$\vec{r}_t^\top A\vec{v}_{t-i} = \frac{1}{\alpha_{t-i+1}}\vec{r}_t^\top(\vec{r}_{t-i} - \vec{r}_{t-i+1}).$$

Deux cas sont possibles :
- Si $i > 1$, $\vec{r}_t^\top \vec{r}_{t-i}$ et $\vec{r}_t^\top \vec{r}_{t-i+1}$ sont tous les deux nuls comme nous venons de le montrer dans la première partie de cette preuve. Le terme $\vec{v}_{t-1}^\top A\vec{v}_{t-i}$ est lui aussi nul d'après nos hypothèses de récurrence.
- Si $i = 1$, le premier terme de la somme dans l'équation A.24 vaut $\vec{r}_t^\top A\vec{v}_{t-1} = -\frac{\vec{r}_t^\top \vec{r}_t}{\alpha_t}$. Le deuxième est donné directement par l'équation A.22, et vaut l'opposé du premier.

Ainsi, quelle que soit la valeur de i, $\vec{v}_t^\top A\vec{v}_{t-i} = 0$. □

Remarque

Les vecteurs $\vec{v}_t$ formant une base de $\mathbb{R}^n$, à partir de la n-ème itération de la méthode du gradient conjugué, $\vec{v}_t = 0$.

A.3.7 Méthodes de quasi-Newton

Il est possible que le calcul de la hessienne nécessite beaucoup de ressources. Dans ce cas, on utilise des méthodes, dites *de quasi-Newton*, qui permettent de remplacer l'inverse de la hessienne par une approximation dans la méthode de Newton. Cette approximation est calculée de manière itérative.

Définition 1.12 (Méthode de quasi-Newton) Soient $\mathcal{U} \subseteq \mathbb{R}^n$ et $f : \mathcal{U} \to \mathbb{R}$ une fonction de classe $\mathcal{C}^1$. Étant donnée une *tolérance* $\epsilon > 0$, on appelle *méthode de quasi-Newton* un algorithme prenant la forme suivante :

1. Choisir $\vec{u}$ aléatoirement dans $\mathcal{U}$.
2. Initialiser W : $W^{(0)} = I$, la matrice identité de même dimension que la hessienne.
3. Tant que $||\nabla f(\vec{u})||_2^2 > \epsilon$:
   - Incrémenter t : $t \leftarrow (t+1)$
   - Actualiser $\vec{u}$: $\vec{u}^{\,(t)} = \vec{u}^{\,(t)} - W^{(t-1)}$
   - Actualiser l'approximation W de $\left(\nabla^2 f(\vec{u})\right)^{-1}$

$\vec{u}^{\,(t)}$ est alors une approximation du point de minimum global de f sur $\mathcal{U}$. ■

Pour approcher l'inverse de la hessienne $\nabla^2 f(\vec{u})^{-1}$, on cherche une matrice $W^{(t)}$ qui soit
- symétrique et semi-définie positive ;
- proche de l'approximation précédente, autrement dit telle que $||W^{(t)} - W^{(t-1)}||_F$ soit minimale (ici $||.||_F$ désigne la norme de Frobenius d'une matrice, autrement dit la racine carrée de la somme des carrés de ses entrées) ;
- telle que $W^{(t)}\left(\nabla f(\vec{u}^{\,(t)}) - \nabla f(\vec{u}^{\,(t-1)})\right) = \vec{u}^{\,(t)} - \vec{u}^{\,(t-1)}$, cette dernière condition, appelée *équation de la sécante*, étant directement dérivée d'un développement de Taylor à l'ordre 1 de ∇f en $\vec{u}^{\,(t-1)}$.

Une des méthodes de quasi-Newton les plus utilisées de nos jours, et les plus performantes, est la méthode *BFGS*, ainsi nommée d'après Charles George Broyden, Roger Fletcher, Donald Goldfarb et David Shanno qui l'ont proposée tous les quatre indépendamment en 1970.

Dans cette méthode, l'approximation itérative de l'inverse de la hessienne est donnée par

$$W^{(t)} = W^{(t-1)} - \frac{d_t \delta_t^\top W^{(t-1)} + W^{(t-1)} \delta_t d_t^\top}{\langle \delta_t, d_t \rangle} + \left(1 + \frac{\langle \delta_t, W^{(t-1)} \delta_t \rangle}{\langle \delta_t, d_t \rangle}\right) \frac{d_t d_t^\top}{\langle \delta_t, d_t \rangle} \tag{A.25}$$

où $d_t = \vec{u}^{\,(t)} - \vec{u}^{\,(t-1)}$ et $\delta_t = \nabla f(\vec{u}^{\,(t)}) - \nabla f(\vec{u}^{\,(t-1)})$.

Il en existe une version moins gourmande en mémoire, appelée *L-BFGS* pour *Limited-memory BFGS*, qui évite d'avoir à stocker en mémoire l'intégralité de la matrice $W^{(t)}$.

A.3.8 Algorithme du gradient stochastique

Dans le cas où n est très grand, le gradient lui-même peut devenir coûteux à calculer. Cependant, on rencontre souvent en apprentissage statistique des fonctions à minimiser qui peuvent se décomposer en une somme de n fonctions plus simples :

$$f(\vec{u}) = \sum_{i=1}^{n} f_i(\vec{u}). \tag{A.26}$$

Son gradient se décompose alors aussi en

$$\nabla f(\vec{u}) = \sum_{i=1}^{n} \nabla f_i(\vec{u}). \tag{A.27}$$

Exemple

C'est le cas par exemple lorsque l'on cherche à minimiser une somme de moindres carrés (voir section 5.1.2) : cette somme s'écrit

$$f(\vec{\beta}) = \sum_{i=1}^{n} \left(y^i - \phi(\vec{x}^{\,i}|\vec{\beta})\right)^2$$

où ϕ est la fonction de prédiction.

L'algorithme du gradient stochastique permet alors d'accélérer les calculs en n'utilisant à chaque itération qu'une seule des fonctions f_i : on remplace $\sum_{i=1}^{n} \nabla f_i(\vec{u})$ par $\nabla f_k(\vec{u})$.

Définition 1.13 (Algorithme du gradient stochastique) Soient $\mathcal{U} \subseteq \mathbb{R}^n$ et $f : \mathcal{U} \to \mathbb{R}$ une fonction de classe $\mathcal{C}^1$, décomposable sous la forme

$$f(\vec{u}) = \sum_{i=1}^{n} f_i(\vec{u}). \tag{A.28}$$

Étant donnés un *pas* $\alpha > 0$ et une *tolérance* $\epsilon > 0$, on appelle *algorithme du gradient stochastique* l'algorithme suivant :

1. Choisir $\vec{u}$ aléatoirement dans $\mathcal{U}$.
2. Tant que $||\nabla f(\vec{u})||_2^2 > \epsilon$:
   — Choisir k aléatoirement parmi $\{1, 2, \ldots, n\}$.
   — Actualiser $\vec{u}$: $\vec{u} \leftarrow \vec{u} - \alpha \nabla f_k(\vec{u})$.

$\vec{u}$ est alors une approximation du point de minimum global de f sur $\mathcal{U}$. ■

On peut bien sûr utiliser aussi un pas adaptatif dans ce contexte.

A.3.9 Descente de coordonnées

Les techniques que nous avons vues ci-dessus supposent que la fonction à minimiser est différentiable. Une technique similaire peut être utilisée pour une fonction non différentiable, mais qui peut se décomposer comme une somme d'une fonction convexe de classe $\mathcal{C}^1$ et de n fonctions convexes chacune d'une seule des coordonnées. On minimise alors la fonction coordonnée par coordonnée.

Définition 1.14 (Descente de coordonnées) Soit $\mathcal{U} \subseteq \mathbb{R}^n$ et $f : \mathcal{U} \to \mathbb{R}$ une fonction de la forme

$$f : \vec{u} \mapsto g(\vec{u}) + \sum_{i=1}^{n} h_i(u_i),$$

où g est une fonction convexe de classe $\mathcal{C}^1$ et les n fonctions h_i sont convexes.

On appelle *descente de coordonnées*, ou *coordinate descent* en anglais, l'algorithme suivant :

1. Choisir $\vec{u}$ aléatoirement dans $\mathcal{U}$.
2. Tant que $||\nabla f(\vec{u})||_2^2 > \epsilon$, actualiser $\vec{u}$:
   - $u_1^{(t)}$ point minimal de $u \mapsto f(u, u_2^{(t-1)}, \dots, u_n^{(t-1)})$
   - $u_2^{(t)}$ point minimal de $u \mapsto f(u_1^{(t-1)}, u, \dots, u_n^{(t-1)})$
   - $\dots$
   - $u_n^{(t)}$ point minimal de $u \mapsto f(u_1^{(t-1)}, u_2^{(t-1)}, \dots, u)$.

$\vec{u}$ est alors une approximation du point de minimum global de f sur $\mathcal{U}$. ■

A.4 Optimisation convexe sous contraintes

Dans cette partie, nous nous intéressons aux problèmes d'optimisation de la forme

$$(\mathcal{P}) : \begin{array}{ll} \min_{\vec{u} \in \mathcal{U}} & f(\vec{u}) \text{ sous les contraintes} \\ & g_i(\vec{u}) \leq 0 \quad \forall i = 1, \dots, m \\ & h_j(\vec{u}) = 0 \quad \forall j = 1, \dots, r, \end{array} \tag{A.29}$$

où les fonctions f, g_i, h_j sont supposées à valeurs réelles et de classe $\mathcal{C}^1$.

De nombreuses méthodes permettent de résoudre ce type de problèmes. Nous détaillerons dans ce qui suit la méthode des multiplicateurs de Lagrange.

A.4.1 Lagrangien

Pour résoudre $(\mathcal{P})$, nous allons introduire son *lagrangien*.

Définition 1.15 (Lagrangien) Soit $(\mathcal{P})$ un problème de minimisation sous contraintes de la forme A.29. On appelle *lagrangien* de $(\mathcal{P})$ la fonction

$$\begin{aligned} L : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^r &\to \mathbb{R} \\ \vec{u}, \vec{\alpha}, \vec{\beta} &\mapsto f(\vec{u}) + \sum_{i=1}^{n} \alpha_i \, g_i(\vec{u}) + \sum_{j=1}^{r} \beta_j \, h_j(\vec{u}). \end{aligned}$$

Les variables $\alpha_1, \alpha_2, \dots, \alpha_m, \beta_1, \beta_2, \dots, \beta_r$ sont appelées *multiplicateurs de Lagrange* ou *variables duales*.

■

Le lagrangien nous permet de définir une fonction, appelée *fonction duale de Lagrange*, dont le maximum donne une borne inférieure à la solution de $(\mathcal{P})$.

Définition 1.16 (Fonction duale de Lagrange) Soient $(\mathcal{P})$ un problème de minimisation sous contraintes de la forme A.29, et L son lagrangien. On appelle *fonction duale de Lagrange* la fonction

$$\begin{aligned} Q : \mathbb{R}^m \times \mathbb{R}^r &\to \mathbb{R} \\ \vec{\alpha}, \vec{\beta} &\mapsto \inf_{\vec{u}\in\mathcal{U}} L(\vec{u}, \vec{\alpha}, \vec{\beta}), \end{aligned}$$

l'infimum $\inf_{\vec{u}\in\mathcal{U}} L(\vec{u}, \vec{\alpha}, \vec{\beta})$ de L en $\vec{u}$ étant la plus grande valeur $q^* \in \mathbb{R}$ telle que $q^* \leq L(\vec{u}, \vec{\alpha}, \vec{\beta})$ pour tout $u \in \mathcal{U}$, et pouvant potentiellement valoir $-\infty$. ■

La fonction duale de Lagrange a l'intérêt d'être une fonction concave, indépendamment de la convexité de $(\mathcal{P})$.

Théorème 1.6 *La fonction duale de Lagrange d'un problème de minimisation sous contraintes de la forme A.29 est concave.* ■

Démonstration. Soient $(\vec{\alpha}_1, \vec{\beta}_1)$, $(\vec{\alpha}_2, \vec{\beta}_2)$ et $0 \leq \lambda \leq 1$.
Prenons $\vec{\alpha} = \lambda\vec{\alpha}_1 + (1-\lambda)\vec{\alpha}_2$ et $\vec{\beta} = \lambda\vec{\beta}_1 + (1-\lambda)\vec{\beta}_2$.
Alors

$$Q(\vec{\alpha}, \vec{\beta}) = \inf_{\vec{u}\in\mathcal{U}} f(\vec{u}) + \vec{\alpha}^{\top}\vec{g} + \vec{\beta}^{\top}\vec{h},$$

où $\vec{g} = (g_1(\vec{u}), g_2(\vec{u}), \dots g_m(\vec{u}))$ et $\vec{h} = (h_1(\vec{u}), h_2(\vec{u}), \dots h_r(\vec{u}))$. Ainsi,

$$\begin{aligned} Q(\vec{\alpha}, \vec{\beta}) &= \inf_{\vec{u}\in\mathcal{U}} f(\vec{u}) + \lambda\vec{\alpha}_1^{\top}\vec{g} + (1-\lambda)\vec{\alpha}_2^{\top}\vec{g} + \lambda\vec{\beta}_1^{\top}\vec{h} + (1-\lambda)\vec{\beta}_2^{\top}\vec{h} \\ &= \inf_{\vec{u}\in\mathcal{U}} \lambda\left(f(\vec{u}) + \vec{\alpha}_1^{\top}\vec{g} + \vec{\beta}_1^{\top}\vec{h}\right) + (1-\lambda)\left(f(\vec{u}) + \vec{\alpha}_2^{\top}\vec{g} + \vec{\beta}_2^{\top}\vec{h}\right), \end{aligned}$$

cette dernière égalité venant de ce que $f(\vec{u}) = \lambda f(\vec{u}) + (1-\lambda)f(\vec{u})$. Enfin,

$$\begin{aligned} Q(\vec{\alpha}, \vec{\beta}) &\geq \lambda \inf_{\vec{u}\in\mathcal{U}} \left(f(\vec{u}) + \vec{\alpha}_1^{\top}\vec{g} + \vec{\beta}_1^{\top}\vec{h}\right) + (1-\lambda) \inf_{\vec{u}\in\mathcal{U}} \left(f(\vec{u}) + \vec{\alpha}_2^{\top}\vec{g} + \vec{\beta}_2^{\top}\vec{h}\right) \\ &\geq \lambda Q(\vec{\alpha}_1, \vec{\beta}_1) + (1-\lambda)Q(\vec{\alpha}_2, \vec{\beta}_2), \end{aligned}$$

et Q est donc concave. □

A.4.2 Dualité faible

La fonction duale de Lagrange d'un problème de minimisation sous contraintes $(\mathcal{P})$ donne une borne inférieure à la solution de $(\mathcal{P})$.

Théorème 1.7 *Soit p^* une solution de $(\mathcal{P})$. Alors quels que soient $\alpha_1, \alpha_2, \dots, \alpha_m \geq 0$ et $\beta_1, \beta_2, \dots, \beta_r \in \mathbb{R}$,*

$$Q(\alpha, \beta) \leq p^*.$$

■

Démonstration. Soit $\vec{u}$ un point admissible. $g_i(\vec{u}) \leq 0$ pour tout $i = 1, \ldots, m$ et $h_j(\vec{u}) = 0$ pour tout $j = 1, \ldots, r$.

Ainsi, $L(\vec{u}, \vec{\alpha}, \vec{\beta}) = f(\vec{u}) + \sum_{i=1}^{n} \alpha_i g_i(\vec{u}) + \sum_{j=1}^{r} \beta_j h_j(\vec{u}) \leq f(\vec{u})$ et donc, pour tout point admissible, $L(\vec{u}, \vec{\alpha}, \vec{\beta}) \leq f(\vec{u})$. On en déduit que

$$\inf_{\vec{u} \in \mathbb{R}^n} L(\vec{u}, \vec{\alpha}, \vec{\beta}) \leq \min_{\vec{u} \in \mathbb{R}^n} f(\vec{u}).$$

□

Puisque $Q(\alpha, \beta) \leq p^*$ pour tout $\vec{\alpha} \in \mathbb{R}_+^m, \vec{\beta} \in \mathbb{R}^r$, le maximum de Q sur $\mathbb{R}_+^m \times \mathbb{R}^r$ est la meilleure de ces bornes inférieures de p^*.

Définition 1.17 (Problème dual de Lagrange) Soient $(\mathcal{P})$ un problème de minimisation sous contrainte de la forme A.29, L son lagrangien, et Q sa fonction duale de Lagrange. On appelle *problème dual de Lagrange* le problème d'optimisation suivant :

$$(\mathcal{Q}) : \quad \begin{array}{cc} \max_{\vec{\alpha} \in \mathbb{R}^m, \vec{\beta} \in \mathbb{R}^r} & Q(\vec{\alpha}, \vec{\beta}) \\ \text{sous les contraintes} \alpha_i \geq 0 & \forall i = 1, \ldots, m \end{array}$$

Le problème $(\mathcal{P})$ est alors appelé *problème primal.*

Enfin, les points de maximisation $\vec{\alpha}^*, \vec{\beta}^*$ sont appelés *multiplicateurs de Lagrange optimaux* ou *variables duales optimales.* ■

Comme Q est concave, le problème dual d'un problème d'optimisation sous contraintes quelconque est un problème d'optimisation sous contraintes *convexe.*

Théorème 1.8 (Dualité faible) *Soient $(\mathcal{P})$ un problème de minimisation sous contrainte de la forme A.29 et $(\mathcal{Q})$ son dual de Lagrange. Appelons d^* une solution de $(\mathcal{Q})$, et p^* une solution du primal $(\mathcal{P})$. Alors*

$$d^* \leq p^*.$$

C'est ce que l'on appelle la dualité faible. ■

A.4.3 Dualité forte

Si la solution du dual minore la solution du primal, elles ne sont pas égales dans le cas général. Pour certaines classes de fonctions, il est possible cependant que $d^* = p^*$. On parle alors de *dualité forte*.

Définition 1.18 (Dualité forte) Soient $(\mathcal{P})$ un problème de minimisation sous contrainte de la forme A.29 et $(\mathcal{Q})$ son dual de Lagrange. Appelons d^* la solution de $(\mathcal{Q})$, et p^* la solution du primal $(\mathcal{P})$. On parle de *dualité forte* si

$$d^* = p^*.$$

■

La *condition de Slater* est une condition suffisante (mais non nécessaire) pour garantir la dualité forte, que l'on doit à Morton L. Slater (1950).

Théorème 1.9 (Condition de Slater) *Soit $(\mathcal{P})$ un problème de minimisation sous contrainte de la forme A.29. Si $(\mathcal{P})$ est convexe, c'est-à-dire f, $g_1, g_2, \ldots, g_m$ sont convexes et $h_1, h_2, \ldots, h_r$ sont convexes, et qu'il existe au moins un point admissible pour lequel les contraintes d'inégalités g_i non affines soient vérifiées strictement, alors la dualité forte est garantie.* ■

Ainsi, on peut résoudre un grand nombre de problèmes d'optimisation convexe sous contraintes en passant par le dual, dont les contraintes ($\alpha_i \geq 0$) sont simples à prendre en compte. On peut en particulier utiliser des méthodes de gradient projeté, qui consistent à « ramener » les itérations successives d'un algorithme à directions de descente dans le domaine des contraintes en les projetant sur cet ensemble.

A.4.4 Conditions de Karush-Kuhn-Tucker

Dans un cadre de dualité forte, si les fonctions f, g_i et h_j sont toutes différentiables, William Karush (dans sa thèse de master de 1939, non publiée) puis Harold W. Kuhn et Albert W. Tucker (Kuhn et Tucker, 1951) ont développé un ensemble de conditions suffisantes à l'optimalité de la solution d'un problème d'optimisation sous contraintes.

Théorème 1.10 (Conditions de Karush-Kuhn-Tucker) *Soit $(\mathcal{P})$ un problème de minimisation sous contrainte de la forme A.29 et $(\mathcal{Q})$ son problème dual.*

Soit un triplet $(\vec{u}^*, \vec{\alpha}^*, \vec{\beta}^*)$ de $\mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^r$ qui vérifie les conditions suivantes, dites de conditions de Karush-Kuhn-Tucker ou conditions KKT :

- *Admissibilité primale : $g_i(\vec{u}^*) \leq 0$ pour tout $i = 1, \ldots, m$ et $h_j(\vec{u}^*) = 0$ pour tout $j = 1, \ldots, r$;*
- *Admissibilité duale : $\alpha_i^* \geq 0$ pour tout $i = 1, \ldots, m$;*
- *Complémentarité des contraintes : $\alpha_i^* g_i(\vec{u}^*) = 0$ pour tout $i = 1, \ldots, m$;*
- *Stationnarité : $\nabla f(\vec{u}^*) + \sum_{i=1}^m \alpha_i^* \nabla g_i(\vec{u}^*) + \sum_{j=1}^r \beta_i^* \nabla h_j(\vec{u}^*) = 0$.*

Alors $\vec{u}^*$ est un point de minimisation du primal $(\mathcal{P})$ et $(\vec{\alpha}^*, \vec{\beta}^*)$ est un point de maximisation du dual $(\mathcal{Q})$. ■

Démonstration. Soit un triplet $(\vec{u}^*, \vec{\alpha}^*, \vec{\beta}^*)$ qui vérifie les conditions de KKT. La condition d'admissibilité primale implique que $\vec{u}^*$ est admissible.

La fonction $\vec{u} \mapsto L(\vec{u}, \vec{\alpha}^*, \vec{\beta}^*)$ est convexe en $\vec{u}$. En effet, $L(\vec{u}, \vec{\alpha}^*, \vec{\beta}^*) = f(\vec{u}) + \sum_{i=1}^n \alpha_i^* g_i(\vec{u}) + \sum_{j=1}^r \beta_j^* h_j(\vec{u})$, les fonctions f et g_i sont convexes, les α_i^* sont positifs, et les h_j sont affines. La condition de stationnarité implique donc que $\vec{u}^*$ minimise $L(\vec{u}, \vec{\alpha}^*, \vec{\beta}^*)$.

Par définition de la fonction duale de Lagrange, $Q(\vec{\alpha}^*, \vec{\beta}^*) = L(\vec{u}^*, \vec{\alpha}^*, \vec{\beta}^*) = f(\vec{u}^*)$. En effet, la condition de complémentarité des contraintes implique
$\sum_{i=1}^n \alpha_i^* g_i(\vec{u}^*) = 0$, et celle d'admissibilité primale implique $\sum_{j=1}^r \beta_j^* h_j(\vec{u}^*) = 0$.

Soient p^* la solution de $(\mathcal{P})$ et d^* celle de $(\mathcal{Q})$. Par définition de d^*, $f(\vec{u}^*) = Q(\vec{\alpha}^*, \vec{\beta}^*) \leq d^*$, et par dualité faible, $d^* \leq p^*$. Ainsi, $f(\vec{u}^*) \leq p^*$ et donc $f(\vec{u}^*) = p^*$: $\vec{u}^*$ est un point de minimisation de $(\mathcal{P})$, et toutes les inégalités précédentes sont donc des égalités et, en particulier, $Q(\vec{\alpha}^*, \vec{\beta}^*) = d^*$ et $(\vec{\alpha}^*, \vec{\beta}^*)$ est un point de maximisation de $(\mathcal{Q})$. □

Géométriquement, les conditions de stationnarité et de complémentarité des contraintes peuvent se comprendre comme suit. Considérons un problème d'optimisation convexe sous une unique contrainte d'inégalité :

$$\begin{array}{c} \min_{\vec{u} \in \mathbb{R}^n} \quad f(\vec{u}) \text{ sous la contrainte} \\ g(\vec{u}) \leq 0. \end{array} \tag{A.30}$$

Soit $\vec{u}_0$ un point de minimisation sans contrainte : $\vec{u}_0 = \min_{\vec{u} \in \mathbb{R}^n} f(\vec{u})$.

Deux cas sont possibles :

- Soit $\vec{u}_0$ appartient au domaine des contraintes, et $\vec{u}^* = \vec{u}_0$ est un point de minimisation de $(\mathcal{P})$ (cas 1) ;
- Soit $\vec{u}_0$ n'appartient pas au domaine des contraintes (cas 2).

Dans le premier cas, $\nabla f(\vec{u}^*) = 0$ (puisque $\vec{u}^*$ est solution du problème sans contrainte) et $g(\vec{u}^*) \leq 0$ (puisque $\vec{u}^*$ appartient au domaine des contraintes).

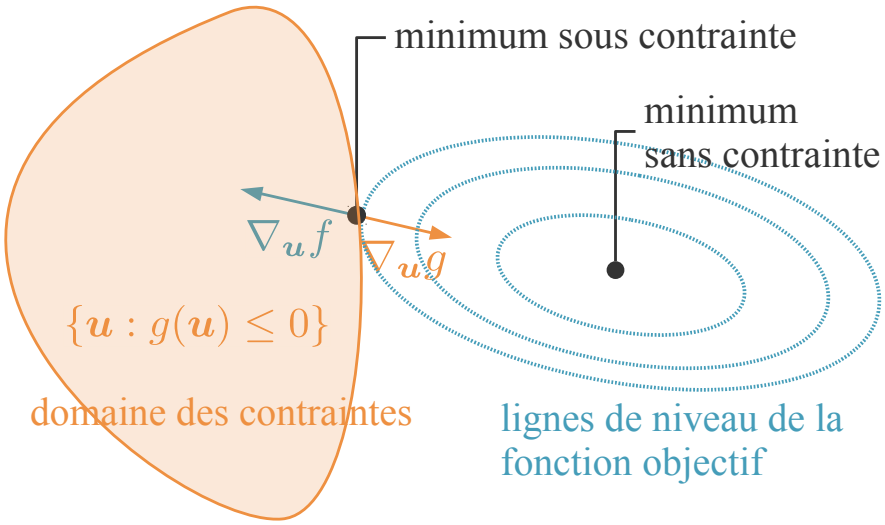


FIGURE A.5 – Le point de minimisation se trouve à la frontière du domaine des contraintes, en un point tangent à une ligne de niveau de f. En ce point, les gradients ∇g et ∇f sont parallèles et de directions opposées.

Dans le deuxième cas, illustré sur la figure A.5, un point de minimisation $\vec{u}^*$ se trouve à la bordure du domaine des contraintes ($g(\vec{u}^*) = 0$), car f étant convexe et minimisée à l'extérieur de ce domaine croît quand on s'en rapproche. Plus précisément, $\vec{u}^*$ se trouve à l'intersection de la bordure du domaine des contraintes et d'une ligne de niveau de f. En ce point, le gradient de f étant tangent à la ligne de niveau et celui de g étant tangent à la bordure du domaine, ∇f et ∇g sont parallèles. De plus, f croît en s'éloignant de $\vec{u}_0$, et g est négative à l'intérieur du domaine des contraintes. ∇f et ∇g sont donc de directions opposées. Ainsi, il existe $\alpha \geq 0$ tel que $\nabla f(\vec{u}^*) = -\alpha \nabla g(\vec{u}^*)$.

Les deux cas peuvent donc être résumés par

$$\nabla f(\vec{u}^*) + \alpha \nabla g(\vec{u}^*) = 0 \text{ et } \alpha g(\vec{u}^*) = 0,$$

où

- soit $\alpha = 0$ et $g(\vec{u}^*) \leq 0$ (cas 1)
- soit $g(\vec{u}^*) = 0$ et $\alpha \geq 0$ (cas 2).

On a ainsi retrouvé la condition de stationnarité $\nabla f(\vec{u}^*) + \alpha \nabla g(\vec{u}^*) = 0$ et celle de complémentarité des contraintes $\alpha g(\vec{u}^*) = 0$.

A.4.5 Programmes quadratiques

Un cas particulier de problème d'optimisation convexe sous contraintes qui revient souvent en machine learning est celui de l'*optimisation quadratique convexe*, ou *convex quadratic programming* (*convex QP*) en anglais. Dans ce cadre, la fonction f à minimiser est quadratique et les contraintes sont affines.

Définition 1.19 Étant donnés n, m, r trois entiers positifs, une matrice semi-définie positive $Q \in \mathbb{R}^{n \times n}$, des vecteurs $\vec{a}, \vec{b}_1, \ldots, \vec{b}_m, \vec{c}_1, \ldots, \vec{c}_r$ de $\mathbb{R}^n$, et des constantes $d, k_1, \ldots, k_m, l_1, \ldots, l_r \in \mathbb{R}$, on ap-

pelle *problème d'optimisation quadratique convexe* un problème de la forme

$$\begin{aligned} \min_{\vec{u}\in\mathbb{R}^n} \ & \frac{1}{2}\vec{u}^{\top} Q\vec{u} + \vec{a}^{\top}\vec{u} + d \text{ sous les contraintes} \\ & \vec{b_i}^{\top}\vec{u} + k_i \leq 0 \quad \forall i = 1,\dots,m \\ & \vec{c_j}^{\top}\vec{u} + l_j = 0 \quad \forall j = 1,\dots,r. \end{aligned}$$

■

Il s'agit d'un problème d'optimisation convexe, vérifiant les conditions de Slater, et dont la région admissible est un polyèdre.

Il existe de nombreuses méthodes pour résoudre ce genre de problèmes, comme les méthodes de point intérieur ou les méthodes d'activation (*active set* en anglais.) De nombreux logiciels et librairies implémentent ces approches, comme CPLEX, CVXOPT, CGAL et bien d'autres.

Pour aller plus loin

- L'ouvrage de Boyd et Vandenberghe (2004) fait figure de référence sur l'optimisation convexe. Pour les aspects numériques, on pourra se reporter à Bonnans et al. (1997) ou à Nocedal et Wright (2006).
- Pour en savoir plus sur les algorithmes à directions de descente stochastique, on pourra se reporter à l'article de Léon Bottou (2012).
- Pour plus de détails sur la dualité de Lagrange, on pourra aussi se reporter au chapitre 8 de Luenberger (1969) ou à Bertsekas et al. (2003).

Bibliographie

Bertsekas, D. P., Nedi; A., et Ozdaglar, A. E. (2003). *Convex Analysis and Optimization*. Athena Scientific, Belmont, MA.

Bonnans, J., Gilbert, J., Lemaréchal, C., et Sagastizábal, C. (1997). *Optimisation numérique : aspects théoriques et pratiques*. Sprinver-Verlag, Berlin Heidelberg.

Bottou, L. (2012). Stochastic gradient descent tricks. `http://leon.bottou.org/publications/pdf/tricks-2012.pdf`.

Boyd, S. et Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press. `https://web.stanford.edu/~boyd/cvxbook/`.

Hestenes, M. et Stiefel, E. (1952). Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 19(6).

Kuhn, H. W. et Tucker, A. W. (1951). Nonlinear programming. In Neyman, J., editor, *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492. University of California Press.

Luenberger, D. G. (1969). *Optimization by Vector Space Methods*. John Wiley & Sons.

Nocedal, J. et Wright, S. J. (2006). *Numerical Optimization*. Springer, Berlin, New York, 2nd edition.

Slater, M. (1950). Lagrange multipliers revisited. *Cowles Commission Discussion Paper*, number 403.