

Big Data et Intelligence Artificielle pour le Génie des Procédés

Jean-Marc Jean-Marc.Commenge@univ-Lorraine.Fr Commenge

▶ **To cite this version:**

Jean-Marc Jean-Marc.Commenge@univ-Lorraine.Fr Commenge. Big Data et Intelligence Artificielle pour le Génie des Procédés. École d'ingénieur. France. 2020. hal-03107557

HAL Id: hal-03107557
https://hal.univ-lorraine.fr/hal-03107557

Submitted on 12 Jan 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Big Data
et
Intelligence Artificielle
pour le Génie des Procédés

Jean-Marc COMMENGE

Ecole Nationale Supérieure des Industries Chimiques

Table des matières

1 Introduction **1**
- 1.1 Données massives, Mégadonnées, Big Data . . . 1
  - 1.1.1 Caractéristiques générales des Mégadonnées . . . 2
  - 1.1.2 Stockage et bases de données . . . 3
- 1.2 Machine Learning, Deep Learning, Intelligence Artificielle . . . 3
- 1.3 Vue générale : Méthodes et objectifs . . . 4

2 Rappels **7**
- 2.1 Méthodes et grandeurs statistiques . . . 7
  - 2.1.1 Moyenne et variance d'une variable aléatoire . . . 7
  - 2.1.2 Matrice de covariance . . . 8
  - 2.1.3 Coefficient et matrice de corrélation . . . 9
- 2.2 Régression linéaire multiple . . . 10
- 2.3 Qualité d'un modèle, d'une régression . . . 11
  - 2.3.1 Diagrammes de parité . . . 11
  - 2.3.2 Coefficient de détermination . . . 13
- 2.4 Compromis biais-variance ou biais-complexité . . . 13

3 Préparation de données **15**
- 3.1 Visualisation . . . 15
- 3.2 Normalisation de variables . . . 17
  - 3.2.1 Normalisation min-max . . . 17
  - 3.2.2 Variable centrée réduite . . . 17
  - 3.2.3 Passage au logarithme . . . 18
- 3.3 Extraction de caractéristiques, de descripteurs, de "features" . . . 18
- 3.4 Sélection de données d'entrainement, de test et de validation . . . 18
  - 3.4.1 Méthode de validation croisée . . . 19
  - 3.4.2 Méthode Bootstrap . . . 20
- 3.5 Applications en Génie des Procédés . . . 20
  - 3.5.1 Matériaux . . . 20
  - 3.5.2 Détection et diagnostic de fautes et dérives . . . 20
  - 3.5.3 Modélisation d'équipements, d'unités, de procédés . . . 20
  - 3.5.4 Sélection d'équipements, synthèse et design de procédés . . . 20

4 Clustering, Partitionnement **21**
- 4.1 Détection de l'existence de clusters . . . 21
- 4.2 Caractéristiques de clusters . . . 22
  - 4.2.1 Centroïde et médoïde . . . 22
  - 4.2.2 Homégénéité d'un cluster et d'un partitionnement . . . 22
  - 4.2.3 Séparabilité . . . 22
  - 4.2.4 Coefficient de silhouette . . . 23
  - 4.2.5 Positions et formes des clusters . . . 23
- 4.3 Méthode des K-moyennes . . . 23
  - 4.3.1 Choix du nombre de clusters K . . . 24
  - 4.3.2 Forme des clusters obtenus . . . 24
- 4.4 Analyse hiérarchique de clusters . . . 25
  - 4.4.1 Approche agglomérative et dendrogramme . . . 25
  - 4.4.2 Fonctions de liens . . . 26

4.5 Méthodes basées sur la densité . . . 26
4.5.1 Notions de ϵ-voisinage, point intérieur, point frontière et point isolé . . . 27
4.5.2 Algorithme DBSCAN . . . 27
4.6 Propagation d'affinités . . . 28
4.7 Modèles de mélange gaussiens . . . 28
4.8 Mean-shift clustering . . . 29
4.9 Modèles de Markov . . . 30
4.10 Méthodes spectrales . . . 30
4.11 Cartes auto-adaptatives, Réseaux de Kohonen . . . 31
4.12 Applications en Génie des Procédés . . . 31
4.12.1 Chimie . . . 31
4.12.2 Matériaux . . . 31
4.12.3 Détection et diagnostic de fautes et dérives . . . 32
4.12.4 Sélection d'équipements, synthèse et design de procédés . . . 32

5 Réduction de dimensionnalité 33
5.1 Analyse en Composantes Principales . . . 33
5.2 Cartes auto-adaptatives, Réseaux de Kohonen . . . 35
5.3 Réseaux de neurones auto-encodeurs . . . 37
5.4 Positionnement multi-dimensionnel . . . 38
5.5 t-SNE . . . 39
5.6 DPLS - Discriminant Partial Least Squares . . . 39
5.7 Analyse discriminante de Fisher . . . 40
5.8 Factorisation en matrices non-négatives . . . 40
5.9 Projections aléatoires . . . 41
5.10 Plongement local linéaire . . . 41
5.11 Plongement spectral . . . 41
5.12 Applications en Génie des Procédés . . . 42
5.12.1 Recherche de descripteurs . . . 42
5.12.2 Chimie et synthèse . . . 42
5.12.3 Matériaux . . . 42
5.12.4 Simulation de réacteurs et de procédés . . . 42
5.12.5 Détection et diagnostic de fautes et dérives . . . 43

6 Classification 45
6.1 Critères de qualité d'une classification . . . 45
6.1.1 Matrice de confusion . . . 45
6.1.2 Spécificité, sensibilité, erreurs de première et seconde espèce . . . 46
6.1.3 Courbe ROC et surface AUC . . . 46
6.2 Méthode des K plus proches voisins . . . 47
6.3 SVM - Séparateur à Vaste Marge . . . 48
6.3.1 Cas linéairement séparable : SVM à marge rigide . . . 48
6.3.2 Cas non linéairement séparable : SVM à marge souple . . . 49
6.3.3 Cas non-linéaire : SVM à noyau . . . 50
6.4 Arbres de décision . . . 51
6.4.1 Critères d'uniformité des régions . . . 51
6.4.2 Construction d'un arbre . . . 52
6.5 Réseaux de neurones . . . 53
6.6 Méthodes ensemblistes . . . 55
6.6.1 Méthodes parallèles : le bagging . . . 55
6.6.2 Méthodes séquentielles : le boosting . . . 56
6.7 Autres méthodes . . . 57
6.7.1 Cartes auto-adaptatives, réseaux de Kohonen . . . 57
6.7.2 Régression multilinéaire . . . 57
6.7.3 Inférence et prédiction bayesienne . . . 57
6.8 Applications en Génie des Procédés . . . 58
6.8.1 Matériaux . . . 58
6.8.2 Détection et diagnostic de fautes et dérives . . . 58
6.8.3 Sélection d'équipements, synthèse et design de procédés . . . 58

7 Réseaux de neurones **59**
7.1 Neurone formel . . . 59
7.1.1 Fonction de transfert . . . 59
7.1.2 Fonction d'activation . . . 60
7.1.3 Types de sorties d'un neurone à 1 entrée . . . 61
7.1.4 Entraînement d'un neurone formel . . . 62
7.1.5 Fonctions d'erreur, fonctions de coût . . . 63
7.2 Perceptron monocouche . . . 63
7.2.1 Expression . . . 63
7.2.2 Paramètres et hyper-paramètres . . . 64
7.2.3 Forme des réponses atteignables . . . 64
7.2.4 Théorème d'approximation universelle . . . 65
7.3 Réseaux multicouches . . . 66
7.4 Calcul et optimisation des poids : entrainement d'un réseau . . . 67
7.4.1 Limites des méthodes d'optimisation habituelles . . . 67
7.4.2 Entraînement - Validation - Test . . . 68
7.4.3 Logique de l'entrainement par rétropropagation . . . 68
7.4.4 Vue générale et comparaison des méthodes d'entrainement . . . 71
7.5 Post-traitement, exploitation et réduction de réseaux de neurones . . . 71
7.5.1 Analyse de sensibilité par rapport aux entrées . . . 72
7.5.2 Problème de l'explicabilité et de l'interprétabilité d'un réseau . . . 72
7.5.3 Réduction de réseaux . . . 72
7.6 Conseils et heuristiques . . . 72
7.6.1 Choix des réseaux de neurones pour la régression . . . 72
7.6.2 Choix des hyper-paramètres du réseau . . . 73
7.6.3 Autres conseils . . . 74
7.7 Applications en Génie des Procédés . . . 74
7.7.1 Chimie, synthèse, et catalyse . . . 74
7.7.2 Chimie analytique . . . 75
7.7.3 Matériaux . . . 75
7.7.4 Propriétés physiques de composés et de mélanges . . . 75
7.7.5 Modélisation et optimisation de réacteurs et de procédés . . . 76
7.7.6 Détection et diagnostic de fautes et dérives . . . 76
7.7.7 Planification de production . . . 77
7.7.8 Contrôle, commande et régulation de procédés . . . 77
7.7.9 Développements méthodologiques . . . 77

8 Réseaux complexes et Deep Learning **79**
8.1 Réseaux convolutionnels . . . 79
8.1.1 Convolution . . . 80
8.1.2 Max Pooling . . . 80
8.2 Réseaux récurrents . . . 81
8.3 Réseaux auto-encodeurs . . . 82
8.4 Applications en Génie des Procédés . . . 82
8.4.1 Classification de données . . . 82
8.4.2 Chimie et synthèse organique . . . 82
8.4.3 Modélisation et optimisation de procédés . . . 82
8.4.4 Design et synthèse de procédés . . . 83
8.4.5 Rectification et réconciliation de données . . . 83
8.4.6 Détection et diagnostic de fautes et dérives . . . 83
8.4.7 Contrôle, commande et régulation de procédés . . . 83

9 Méthodes en flux **85**
9.1 Construction et actualisation de modèles . . . 85
9.1.1 Régression multilinéaire récursive . . . 85

10 Outils logiciels **87**
10.1 Outils logiciels . . . 87
10.2 Matlab . . . 88
10.2.1 Fonctions et instructions généralistes . . . 88
10.2.2 Variables, vecteurs, tableaux . . . 88
10.2.3 Fonctions mathématiques et statistiques . . . 88
10.2.4 Fonctions graphiques . . . 88
10.2.5 Structures algorithmiques de base . . . 88
10.2.6 Actions sur des fichiers . . . 91
10.2.7 Clustering et partitionnement . . . 91
10.2.8 Réduction de dimensionnalité . . . 96
10.2.9 Classification . . . 96
10.2.10 Réseaux de neurones : Perceptron monocouche . . . 100
10.2.11 Réseaux complexes et Deep Learning . . . 102

Exercices **105**
11.1 Préparation de données . . . 105
11.1.1 Analyse d'une matrice de projections . . . 105
11.1.2 Matrices de covariance et de corrélation . . . 105
11.1.3 Normalisation de données brutes . . . 106
11.1.4 Régression de données temporelles . . . 106
11.1.5 Evolution des parcs régionaux éolien et solaire (2001-2019) . . . 106
11.2 Clustering, partitionnement . . . 108
11.2.1 Clustering par K-moyennes de données régulières . . . 108
11.2.2 Choix d'une méthode de clustering . . . 108
11.2.3 Histogrammes de distances entre données . . . 108
11.2.4 Procédé de carbonatation . . . 108
11.2.5 Etude des hyper-paramètres d'une méthode DBSCAN . . . 110
11.2.6 Profils types d'élèves-ingénieurs . . . 111
11.3 Réduction de dimensionnalité . . . 112
11.3.1 Nombres de composantes principales d'une ACP . . . 112
11.3.2 Récupération de données après réduction de dimensionnalité . . . 112
11.3.3 Réduction de dimensionnalité ? . . . 112
11.3.4 Taille des couches cachées d'un réseau auto-encodeur multicouche . . . 112
11.3.5 ACP sur des données dans $\mathbb{R}^6$ . . . 113
11.3.6 Carte de Kohonen de données dans $\mathbb{R}^3$ . . . 113
11.4 Classification . . . 113
11.4.1 Classification à 2 classes par réseaux de neurones . . . 113
11.4.2 Courbe ROC d'une classification binaire . . . 113
11.4.3 Détection automatique de régimes hydrodynamiques en microcanal . . . 114
11.5 Réseaux de neurones . . . 115
11.5.1 Caractéristiques de perceptrons monocouches . . . 115
11.5.2 Diagnostic des résultats d'entrainement d'un perceptron monocouche . . . 115
11.5.3 Portes logiques . . . 115
11.5.4 Régression de données paraboliques . . . 115
11.5.5 Le Mont Fuji . . . 116

Glossaire, abréviations **117**

Chapitre 1

Introduction

Depuis les années 2000, le développement de l'internet et des technologies de communications ont provoqué l'explosion des quantités de données générées par tous les acteurs de la société. Qu'il s'agisse du commerce 2.0, des réseaux sociaux, des télécommunications, des systèmes financiers, des administrations ou des infrastructures scientifiques et industrielles, des Teraoctets de données nouvelles sont produites chaque jour. Ces mégadonnées ("**Big Data**") ont nécessité d'une part la construction d'équipements dédiés (réseaux, datacenters, bases de données, etc.) et d'autre part le développement de méthodes pour leur traitement, leur analyse, leur compression, etc.

Alors que de nombreuses méthodes de traitement de données existent depuis longtemps, très peu de méthodes étaient adaptées à la gestion de si grandes quantités d'informations et à leur diversité (réels, textes, images, sons, vidéos, flux continu, etc.). De très nombreuses méthodes ont donc été développées, et sont maintenant rassemblées sous le terme "Fouille de données" ("**Data Mining**") : elles permettent de réduire la taille de ces données, d'y chercher des structures, des dépendances internes, des corrélations, des modélisations, etc.

Cette analyse des données massives a mis en évidence la richesse de l'information qu'elles contiennent : plus que l'information portée par les données elles-mêmes, la multiplicité des données et leurs ressemblances et/ou différences renseigne sur des caractéristiques intrinsèques du système qui les génère, bien que ces caractéristiques ne soient pas explicitement quantifiées par ces données. Par exemple, l'analyse des mots-clefs utilisés sur un moteur de recherche renseigne sur la santé d'une population et la possible émergence d'une épidémie (Google Flu Trends). De même, l'analyse des signaux renvoyés par les capteurs d'un procédé industriel renseigne sur son bon fonctionnement ou sur de possibles dérives de certains équipements nécessitant une maintenance (bouchage, encrassement, vieillissement de catalyseurs, etc.).

Néanmoins, cette exploitation utile des données massives est rendue difficile par la complexité des données et de leurs interactions : il est impossible pour un cerveau humain d'appréhender cette complexité pour en extraire des corrélations significatives. On ne sait d'ailleurs même pas très bien quoi chercher, ni comment le formaliser. Cette difficulté a conduit au développement de nouvelles méthodes, qualifiées d'"**Intelligence Artificielle**", capables d'apprendre à construire des relations/corrélations après avoir étudié un grand nombre de données. Cet "Apprentissage automatique", appelé "**Machine Learning**", regroupe des méthodes très variées qui donnent l'impression que le programme apprend et invente alors qu'il construit simplement une corrélation complexe. Ces méthodes ayant démontré leur efficacité sur des problèmes difficiles, elles ont été développées et ont amené à l'"Apprentissage Profond" ("**Deep Learning**") qui résout désormais des problèmes qu'on ne sait pas formaliser (Jeu de Go, voitures autonomes, etc.).

Ce chapitre d'introduction revient sur les caractéristiques des données massives et présente une vue d'ensemble des méthodes d'intelligence artificielle. Les chapitres suivants présentent certaines de ces méthodes utilisées en Génie des Procédés.

1.1 Données massives, Mégadonnées, Big Data

L'exploitation des mégadonnées peut être abordée selon différents angles. La première façon de les présenter est liée à leur forme et leur contenu, qui seront détaillés ci-dessous et qui sont les caractéristiques les plus importantes pour une exploitation en Génie des Procédés. Néanmoins, elles ne peuvent pas être totalement découplées des problèmes de gestion et de stockage qu'elles induisent : ces aspects ne seront présentés que brièvement.

1.1.1 Caractéristiques générales des Mégadonnées

Les caractéristiques des données massives sont classiquement synthétisées selon le "Principe des 3V" qui a été étendu à 5V : volume, variété, vélocité, valeur, véracité.

Volume

La caractéristique "Volume" fait référence à la quantité d'informations acquises, stockées, traitées, analysées et diffusées. Elle correspond au "Big" de "Big Data". La Figure 1.1 présente une échelle des ordres de grandeurs des tailles de données générées par différentes sources. Depuis une simple variable réelle codée dans une mémoire informatique jusqu'à l'ensemble des données transférées sur le réseau internet annuellement, l'échelle s'étend sur 21 ordres de grandeurs. En 2020, la messagerie Gmail stocke l'équivalent d'un milliard de DVD de données de ses utilisateurs, et un DVD est capable à son tour de stocker un milliard de nombres réels en double précision.

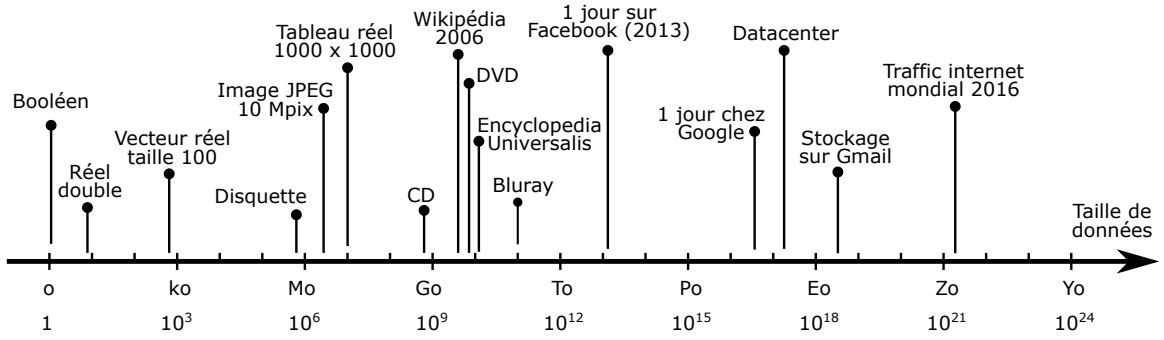


FIGURE 1.1 – Echelle des tailles de données et de supports, en octets. Mo = Mégaoctets, Go = Gigaoctets, To = Téraoctets, Po = Pétaoctets, Eo = Exaoctets, Zo = Zettaoctets, Yo = Yottaoctets.

Variété

La variété des données réside dans l'hétérogénéité de leurs formats et de leurs types. Il peut s'agir de nombres réels, de tableaux, de fichiers numériques, de caractères, de textes, d'images, de sons, de vidéos ou de tout ensemble de ces formats élémentaires. Si on se restreint au Génie des Procédés, la plupart des données issues de capteurs de mesure sont des nombres réels, ou des fichiers historiques rassemblant des données réelles sous formes de tableaux. De plus en plus souvent, ces données sont des images qui feront l'objet d'analyses, notamment pour des problèmes de caractérisation d'échantillons et de surveillance. Il existe des exemples d'applications où les données sont des sons issus de capteurs de vibrations, ou des textes notamment pour l'analyse de scénarios de sécurité.

Vélocité / Vitesse

Le terme "vélocité" fait référence au caractère dynamique ou temporel des données, à leur actualisation en continu sous la forme d'un flux. L'analyse de ces flux peut être effectuée, soit en différé si les données sont stockées et traitées par lots, soit en temps réel lorsque des méthodes de streaming sont disponibles. La Figure 1.2 présente quelques exemples de flux de données. Alors qu'un capteur seul génère un flux très raisonnable, le traitement de milliers de capteurs sur un site industriel génère rapidement des flux importants. Ces flux restent néanmoins acceptables par rapport à d'autres applications, notamment celles liées à internet.

Valeur

La caractéristique "Valeur" d'une donnée n'est pas liée à la valeur que prend cette donnée (x=4, name = "hortensia"), ni à la valeur financière du système qui a été mis en oeuvre pour l'acquérir, mais à son potentiel de valorisation, notamment en termes économiques. Pour une entreprise ou une administration, une meilleure gestion des données internes peut se traduire par des économies et une plus grande efficience de fonctionnement. L'analyse des données d'un marché ou d'un portefeuille de clients permet d'identifier de nouveaux besoins et peut mener à la création de nouveaux produits, donc accroître la compétitivité. En Génie des Procédés, le suivi des données de procédés est utilisé pour anticiper la maintenance d'équipements : cela permet de réduire les temps d'arrêts d'unité pour augmenter la capacité de production annuelle et baisser le coût de revient d'une production.

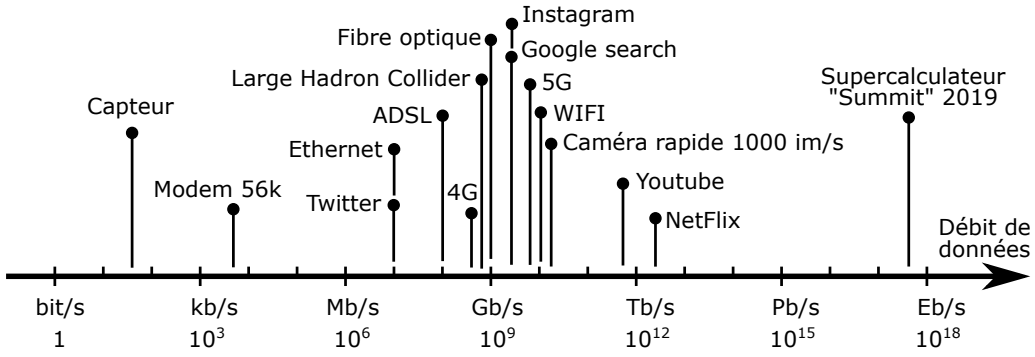


FIGURE 1.2 – Echelle des débits de données, en bit/s.

Véracité / Validité

La véracité d'une donnée fait référence à sa qualité et aux problèmes éthiques liés à son utilisation. Techniquement, cela correspond à la fiabilité qu'on peut accorder à une donnée (valeurs aberrantes, biaisées ou manquantes) et à la confiance qu'on peut avoir dans le système ou la personne qui a généré cette donnée. En Génie des Procédés, la véracité est associée aux problèmes classiques d'erreurs de mesures, d'incertitudes expérimentales, de dérives de capteurs, de réconciliation de données, etc.

1.1.2 Stockage et bases de données

Les données massives soulèvent également des questions techniques et méthodologiques liées à leur gestion et leur stockage. Comme ces aspects ne relèvent pas directement des compétences d'ingénieurs en Génie des Procédés, elles ne sont que très brièvement mentionnées ici. Le lecteur intéressé pourra consulter Espinasse (2017) et Tufféry (2019).

- Cloud computing : la taille des mégadonnées ne permet pas de les stocker sur un seul disque dur, une seule machine ou même un datacenter. Leur stockage distribué est donc préconisé, même si cela ne permet pas d'assurer les propriété ACID (Atomicité, Cohérence, Isolation, Durabilité) de ces données.
- MapReduce : MapReduce est un modèle d'architecture de développement informatique permettant la parallélisation massive et le traitement distribué de données volumineuses. Il repose sur deux fonctions "Map" et "Reduce" : "Map" analyse un problème sur un noeud et le découpe en sous-problèmes alors que "Reduce" traite en parallèle les sous-problèmes sur différents noeuds, qui renvoient finalement le résultat au noeud original.
- Hadoop (High-Availability Distributed Object-Oriented Platform) : Hadoop est un cadriciel libre et open source qui intègre la fonction MapReduce.
- Bases de données NoSQL : le SQL est un langage de recherche sur des bases de données (Structured Query Language) conçu pour exploiter des données structurées. Or les mégadonnées présentent une telle variété de types qu'une évolution du langage a été nécessaire : "NoSQL" signifie "Not Only SQL". Il existe plusieurs types de bases de données NoSQL : les modèles orientés "clé-valeur", les modèles orientés "documents", les modèles orientés "colonnes" et les modèles orientés "graphes".
- NewSQL : le système NewSQL est une nouvelle architecture de stockage de données qui vise à réconcilier le SQL et le NoSQL.

1.2 Machine Learning, Deep Learning, Intelligence Artificielle

L'Intelligence Artificielle (IA) est généralement définie comme l'ensemble des théories et des techniques mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence. Ce domaine regroupe donc un ensemble de concepts et de technologies, notamment numériques, mais ne désigne pas une discipline autonome. Ces méthodes font appel à la neurobiologie informatique (réseaux neuronaux), à la logique mathématique (partie des mathématiques et de la philosophie) et à l'informatique. Les méthodes développées visent à résoudre des problèmes à forte complexité logique ou algorithmique.

Il existe une confusion fréquente entre intelligence artificielle, apprentissage automatique (machine learning) et apprentissage profond (deep learning). Ces notions ne sont pas équivalentes, mais sont imbriquées (Figure 1.3) :

- l'intelligence artificielle englobe l'apprentissage automatique, qui représente beaucoup de méthodes discutées dans la section suivante, telles que la méthode des k-moyennes, les SVM (séparatrices à vaste marge), les arbres de décision, les k plus proches voisins (kNN), et les réseaux de neurones.

— les réseaux de neurones jouent un rôle très particulier dans les domaines de l'apprentissage automatique car ils constituent une méthode adaptée à de nombreux types de problèmes de classification et de régression
— les réseaux de neurones simples sont généralement considérés comme étant de l'apprentissage automatique. Par contre, lorsque le nombre de couches augmente et que les propriétés des couches de neurones sont conçues pour effectuer des tâches de complexité croissante, alors on parle d'apprentissage profond.
— L'apprentissage profond désigne un ensemble de méthodes tentant de modéliser avec un haut niveau d'abstraction des données grâce à des architectures articulées de différentes transformations non linéaires. Les méthodes d'apprentissage profond se basent très majoritairement, mais pas exclusivement, sur des réseaux de neurones multicouches de structures complexes.

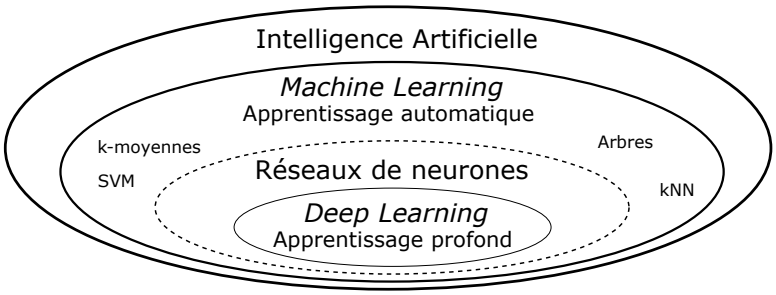


FIGURE 1.3 – Représentation schématique des principaux domaines de l'intelligence artificielle.

1.3 Vue générale : Méthodes et objectifs

Il est assez difficile de se faire une idée claire et précise de l'ensemble des méthodes que représentent le Big Data et l'Intelligence Artificielle. En effet, il s'agit d'un domaine récent, encore en développement permanent, au sein duquel plusieurs communautés interagissent avec leurs jargons et leurs habitudes propres : les conventions des informaticiens, des logiciens et des statisticiens ne sont pas toujours uniformes, ce qui peut mener à des confusions ou des erreurs de catégories.

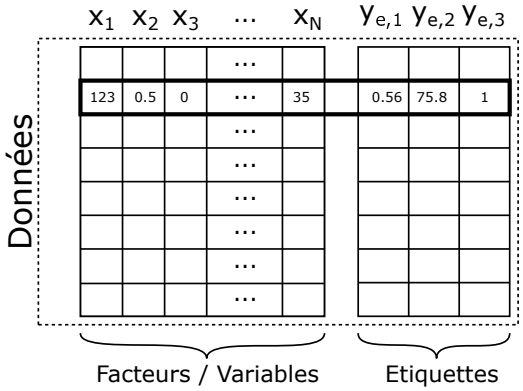


FIGURE 1.4 – Exemple de données.

Par contre, il est impossible de comprendre les particularités et les objectifs des méthodes sans faire référence aux données auxquelles elles s'appliquent. La Figure 1.4 présente un lot de données qu'on souhaite exploiter. Chaque ligne représente une donnée, et l'ensemble des données constitue un tableau. Les premières colonnes, notées x_i correspondent aux variables ou facteurs. Les dernières colonnes, notées y_{e_j}, correspondent aux "sorties" donc aux grandeurs qu'on a mesurées pour caractériser le système : en intelligence artificielle, on les appelle "étiquettes". En génie des procédés, ces données pourraient correspondre à un ensemble de mesures effectuées sur un réacteur de synthèse chimique où les variables pourraient être, par exemple, la température, la concentration, le temps de coulée d'un réactif, etc. et les étiquettes pourraient être la conversion, la sélectivité, le rendement, etc. Ici, chaque case contient un réel ou un entier, mais il pourrait s'agir de variables d'autres types (textes, images, etc.).

Sur un tel tableau de données, les différentes méthodes d'apprentissage automatique vont pouvoir travailler, soit sur les colonnes, soit sur les lignes. Elles peuvent également tenir compte, ou non, des étiquettes : si les

étiquettes sont prises en compte, on parle d'apprentissage "supervisé".

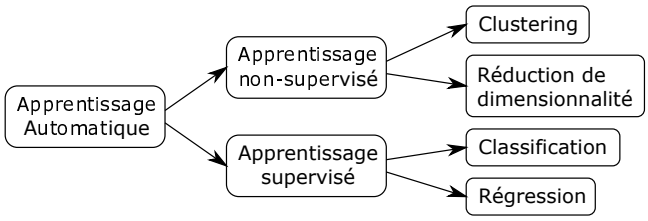


FIGURE 1.5 – Sous-catégories de méthodes d'apprentissage automatique.

Les méthodes d'**apprentissage non-supervisé** ne tiennent pas compte des étiquettes et ne travaillent que sur le tableau des variables des données, les x_i. Dans l'espace des variables x_i qui est de dimension N, chaque ligne de ce tableau représente un point. Les méthodes d'apprentissage non-supervisé étudient la position de ces points dans cet espace et cherchent des structures, des ressemblances ou des clusters. La Figure 1.6 (gauche) présente une centaine de données caractérisées par deux variables x_1 et x_2. Un oeil humain repère tout de suite 3 sous-groupes. Les méthodes de **clustering** visent à trouver de tels groupes dans les données sans savoir a priori s'il en existe, et sans connaître le nombre de clusters s'il y en a. Ici, le clustering a permis d'identifier 3 clusters (Figure 1.6, centre) et d'affecter chaque donnée à l'un de ces clusters en lui attribuant un symbole différent (rond rouge, triangle bleu, ou carré vert). Ces méthodes permettent de distinguer des sous-populations dans les données.

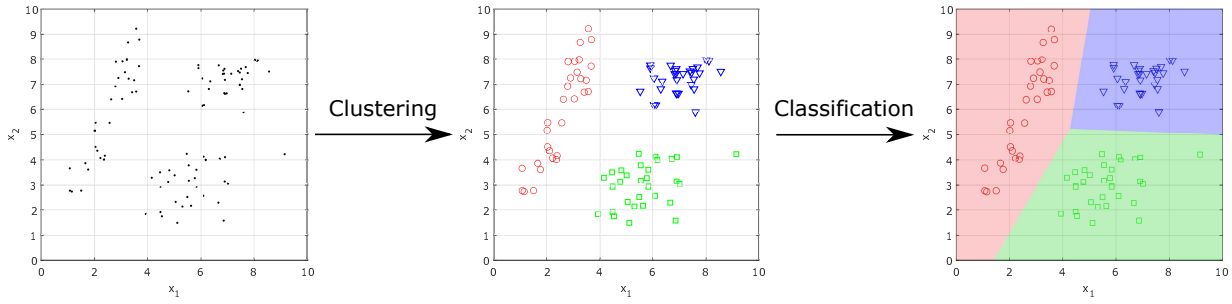


FIGURE 1.6 – Visualisation des étapes de clustering et de classification appliquées à un lot de données non-étiquetées.

En général, les variables x_i sont indépendantes et il n'existe pas, a priori, de relations entre les colonnes de ce tableau : on s'intéresse préférentiellement aux lignes comme expliqué ci-dessus. Néanmoins, selon l'origine des données, il peut exister des relations entre les colonnes. Or de telles relations signifient que les données n'occupent pas tout l'espace des x_i de dimension N mais se trouvent en fait sur un sous-espace de dimension plus faible. Il est donc sans doute possible de réduire la taille des problèmes en travaillant directement dans le sous-espace de dimension faible plutôt que dans l'espace complet : cette **réduction de dimensionnalité** est une caractéristique très utile en Big Data pour faciliter l'exploitation des données massives.

Parmi les méthodes de **clustering**, on peut citer :
— la méthode des k-moyennes,
— la méthode des K plus proches voisins
— les arbres de décision,
— les forêts aléatoires,
— l'inférence bayesienne naïve,
— l'analyse hiérarchique de clusters,
— les modèles à chaîne de Markov cachées.
Parmi les méthodes de **réduction de dimensionnalité**, on peut citer :
— l'analyse en composantes principales (ACP),
— les réseaux de neurones autoencodeurs.

Les méthodes d'**apprentissage supervisé** tiennent compte des étiquettes des données, $y_{e,j}$, mais on distingue deux utilisations de ces étiquettes :
— si les étiquettes sont des valeurs discrètes (binaires 0 ou 1 ; entières 1, 2, 3, 4, etc.), elles ne servent généralement qu'à distinguer des classes de données (issues de l'expérience ou d'une étape préalable

de clustering). On parle alors de **classification**, de "mapping" et les méthodes correspondantes vont permettre de modéliser les frontières entre ces classes en fonction des variables x_i. La Figure 1.6 (droite) montre les frontières entre domaines proposées par une méthode de classification appliquée aux données préalablement clusterisées. Ces méthodes permettent ensuite de classer rapidement toute nouvelle donnée acquise vers l'un des 3 sous-groupes représentés par les 3 clusters, simplement en fonction des valeurs de ses variables x_1 et x_2.

— si les étiquettes sont des valeurs réelles continues, l'objectif consiste généralement à trouver une relation mathématique explicite entre les x_i et une (ou des) étiquette $y_{e,j}$: il s'agit d'un problème de **régression**, de modélisation mathématique. La Figure 1.7 illustre l'application d'une méthode de régression à des données constituées de 2 variables $0 \leq x_1 \leq 10$ et $0 \leq x_2 \leq 10$ et d'un étiquette réelle y dont les valeurs sont situées entre 0 et environ 4.5 (gauche). Sur la figure de droite, les données sont superposées aux courbes isovaleurs estimées par un modèle de régression. Cette régression permet d'interpoler entre les données sur l'ensemble du domaine. Selon les méthodes, les expressions mathématiques des régressions peuvent être simples ou très complexes, et fournies par l'utilisateur ou construites par la méthode elle-même : c'est notamment cette dernière caractéristique qui fait le succès des réseaux de neurones artificiels.

Parmi les méthodes de **classification**, on peut citer :
— les séparateurs à vaste marge (SVM, Support Vector Machine),
— les arbres de décision,
— les forêts aléatoires,
— les réseaux de neurones artificiels.

Parmi les méthodes de **régression**, on peut citer :
— la régression multilinéaire,
— la régression ridge,
— la régression logistique,
— la régression par krigeage,
— la régression par processus gaussiens,
— les réseaux de neurones artificiels.

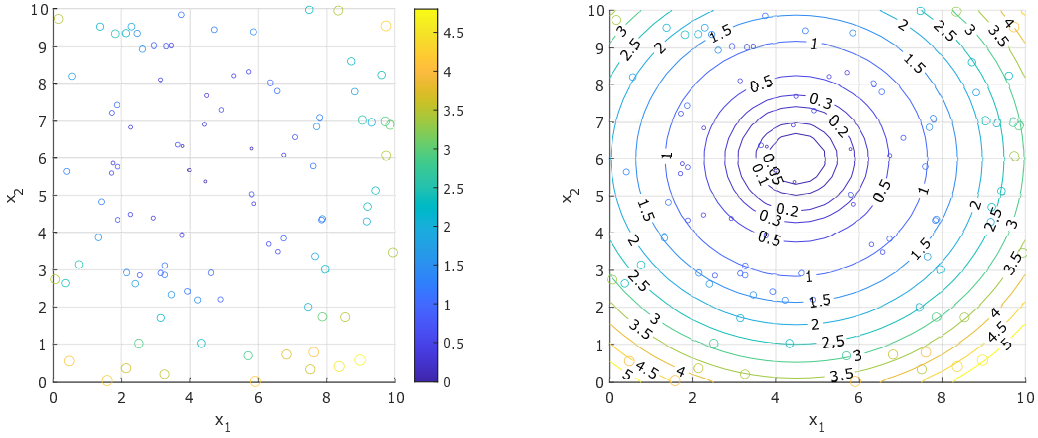


FIGURE 1.7 – Illustration d'une méthode de régression appliquée à des données constituées de 2 variables x_1 et x_2 et d'un étiquette réelle y.

Chapitre 2

Rappels

Ce chapitre rappelle quelques notions fondamentales de statistiques qui sont réutilisées par les méthodes de fouille de données et d'apprentissage automatique. Ces notions ne sont ici que résumées, une description plus détaillée est disponible dans le support de cours "Méthodes statistiques" (ENSIC, 1A, I_2C).

2.1 Méthodes et grandeurs statistiques

2.1.1 Moyenne et variance d'une variable aléatoire

Soit un échantillon de N observations x_i de la variable aléatoire x obéissant à une densité de probabilité (Figure 2.1).

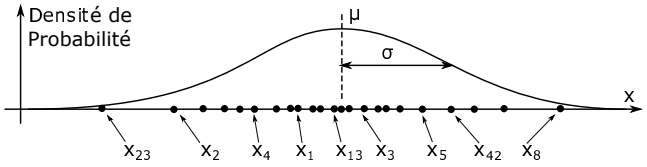


FIGURE 2.1 – Illustration d'un échantillonnage d'une variable x aléatoire.

On peut estimer la moyenne vraie μ de cette distribution en calculant une moyenne d'échantillonnage, $\bar{x}$, à partir des N observations x_i :

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{2.1}$$

La moyenne d'échantillonnage, $\bar{x}$, n'est qu'une estimation ponctuelle de la moyenne vraie μ, mais c'est une estimation non biaisée puisque l'espérance de $\bar{x}$ est égale à μ : $E(\bar{x}) = \mu$: si la taille de l'échantillon, N tend vers l'infini, $\bar{x}$ tend vers μ.

On peut également obtenir une estimation de l'écart-type, σ, de la densité de probabilité, à savoir grossièrement sa demi-largeur à mi-hauteur (pour les distributions quasi-gaussiennes), en calculant l'écart-type d'échantillonnage, s_x :

$$s_x = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N-1}} \tag{2.2}$$

La variance d'échantillonnage, s_x^2, qui est aussi une estimation de la variance vraie σ_x^2, n'est rien d'autre que le carré de l'écart-type :

$$s_x^2 = \frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N-1} \tag{2.3}$$

La forme de cette expression est importante puisque plusieurs expressions peuvent être utilisées. Ces expressions se distinguent par le numérateur, N, $N-1$ ou $N-2$ selon les cas, et sont toutes des estimateurs non-biaisés de la variance vraie, mais sont des estimateurs plus ou moins efficaces. Dans l'expression ci-dessus, le numérateur est une somme de carrés d'écarts et le dénominateur est le nombre de degrés de libertés associés

à cette somme de carrés : comme les x_i sont N variables aléatoires, elles introduisent N degrés de liberté, or le calcul de $\bar{x}$ a ajouté une équation et supprimé un degré de liberté, donc il reste $N-1$ degrés de libertés associés à cette somme. L'utilisation de cette expression est souvent plus explicative que celles avec N ou $N-2$, notamment en analyse de variance où le théorème de partition s'applique aux sommes de carrés et aux degrés de liberté.

2.1.2 Matrice de covariance

Lorsque les données se trouvent dans un espace à plusieurs dimensions (points $(x;y)$ en 2 dimensions, points $(x_1;x_2;x_3)$ en 3 dimensions), la notion de moyenne reste similaire à son équivalent monodimensionnel : $\bar{x}$, $\bar{y}$, $\bar{x}_1$, $\bar{x}_2$, etc.

Par contre, la notion de variance se complexifie et s'enrichit par l'idée de covariance. Pour la suite, on étudiera un échantillon, ou un ensemble de points, dans un espace à p dimensions. Chaque donnée/point k est caractérisé par des facteurs/coordonnées de la forme $(x_{1,k};x_{2,k};x_{3,k};...x_{p,k})$. Si on range ces données sous forme de lignes dans un tableau, chaque colonne p contient un vecteur X_p qui recense toutes les coordonnées des données selon la dimension p : X_p est la projection de l'ensemble des données sur l'axe p. Pour chaque dimension X_p, on peut calculer une variance comme précédemment. Ces différentes variances renseignent sur la dispersion du nuage de points dans l'espace.

La notion de covariance permet de tester si deux variables/coordonnées X_i et X_j varient simultanément ou sont totalement indépendantes l'une de l'autre. La covariance est calculable telle que :

$$Cov(X_i,X_j) = Cov(X_j,X_i) = \sigma_{x_ix_j} = \frac{1}{N-1}\sum_{k=1}^{N}(x_{i,k}-\bar{x_i})(x_{j,k}-\bar{x_j}) \tag{2.4}$$

On remarque, d'après cette formule, que la covariance d'une variable par rapport à elle-même est égale à sa propre variance. La covariance entre deux variables est nulle si les deux variables sont indépendantes. Comme i et j peuvent prendre toutes les valeurs entre 1 et p, toutes ces covariances sont stockées dans une matrice de covariance à p lignes et p colonnes, habituellement notée $\mathbf{\Sigma}$:

$$\mathbf{\Sigma} = \begin{pmatrix} \mathrm{Var}(X_1) & \mathrm{Cov}(X_1,X_2) & \cdots & \mathrm{Cov}(X_1,X_p) \\ \mathrm{Cov}(X_2,X_1) & \ddots & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Cov}(X_p,X_1) & \cdots & \cdots & \mathrm{Var}(X_p) \end{pmatrix} = \begin{pmatrix} \sigma^2_{x_1} & \sigma_{x_1x_2} & \cdots & \sigma_{x_1x_p} \\ \sigma_{x_2x_1} & \ddots & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{x_px_1} & \cdots & \cdots & \sigma^2_{x_p} \end{pmatrix} \tag{2.5}$$

Il est important de remarquer que la covariance peut être négative, d'où la notation $\sigma_{x_ix_j}$ sans le "carré" habituellement associé à une variance $\sigma^2_{x_1}$. Il ne faut tout de même pas la confondre avec un écart-type. Il faut aussi remarquer que la matrice de covariance est symétrique, par définition même de la covariance : $\sigma_{x_ix_j} = \sigma_{x_jx_i}$.

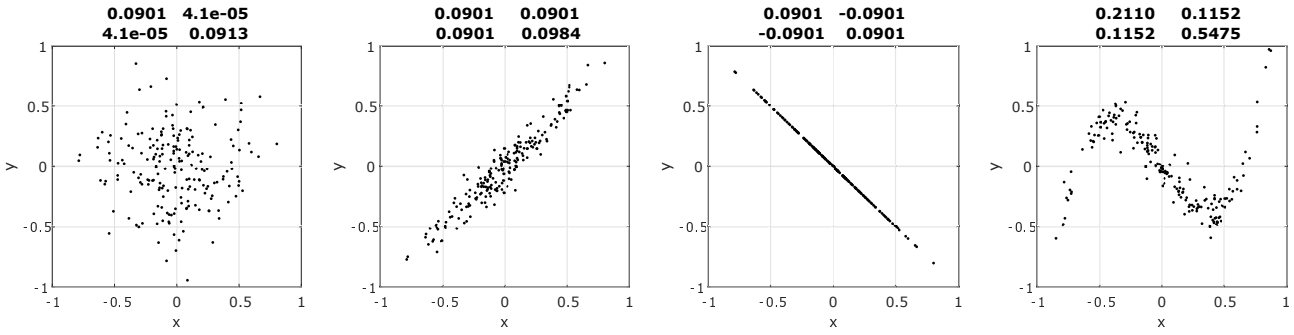


FIGURE 2.2 – Exemples de données de degrés de corrélation variables et matrices de covariance associées.

Cette matrice de covariance contient une information très valorisable pour mieux comprendre les données en grande dimension. La Figure 2.2 présente 4 exemples de données à 2 dimensions $(x;y)$:

— sur la première figure, les deux composantes x et y des données sont totalement décorrélées : y n'est pas lié à x et réciproquement. La matrice de covariance possède donc des termes hors diagonale quasiment nuls. Les deux variances des x et y valent environ 0.09, soit un écart-type proche de 0.3, ce qui semble cohérent avec la disposition des points dans le plan.

— sur la seconde figure, on observe que x et$y$ sont corrélés positivement : y augmente avec x. Les covariances hors-diagonales sont non-nulles et positives.

— sur la troisième figure, y est parfaitement corrélé en fonction de x, le points s'alignent sur la seconde bissectrice $y = -x$. Cette corrélation négative se traduit par des covariances hors-diagonales négatives. On remarque de plus qu'au signe près, les 4 valeurs de la matrice sont égales. Cela se démontre facilement en faisant appel à deux propriétés des variances et covariances :

$$Var(a.X) = a^2.Var(X) \qquad \text{et} \qquad Cov(c.X, Y) = c.Cov(X, Y) \tag{2.6}$$

— sur la dernière figure, on observe que x et y semblent corrélés par une relation polynomiale de degré 3. La matrice de covariance ne renseigne pas sur le type de couplage entre variables, mais elle montre qu'il existe une relation et que les deux variables ne sont pas indépendantes l'une de l'autre.

2.1.3 Coefficient et matrice de corrélation

Pour des donnnées x_i et y_i $(1 \leq i \leq n)$, on définit le coefficient de corrélation entre les x_i et les y_i tel que :

$$r_{xy} = r_{yx} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} = \frac{\sigma_{x_i y_i}}{s_{x_i} s_{y_i}} \tag{2.7}$$

Ce coefficient est égal au rapport de la covariance entre les x_i et les y_i sur les écart-types respectifs des x_i et des y_i. Ce coefficient informe sur la possible existence de corrélations linéaires entre les deux séries de données. La Figure 2.3 présente plusieurs jeux de données, possèdant des degrés de corrélation variés, et leurs coefficients de corrélation associés. Il faut noter tout d'abord que ce coefficient peut être positif ou négatif, mais il est compris entre -1 et +1 : une valeur positive indique que les x_i et les y_i augmentent/diminuent simultanément, alors qu'une valeur négative induit une décroissance de l'un lorsque l'autre augmente. On parle de corrélation positive ou négative.

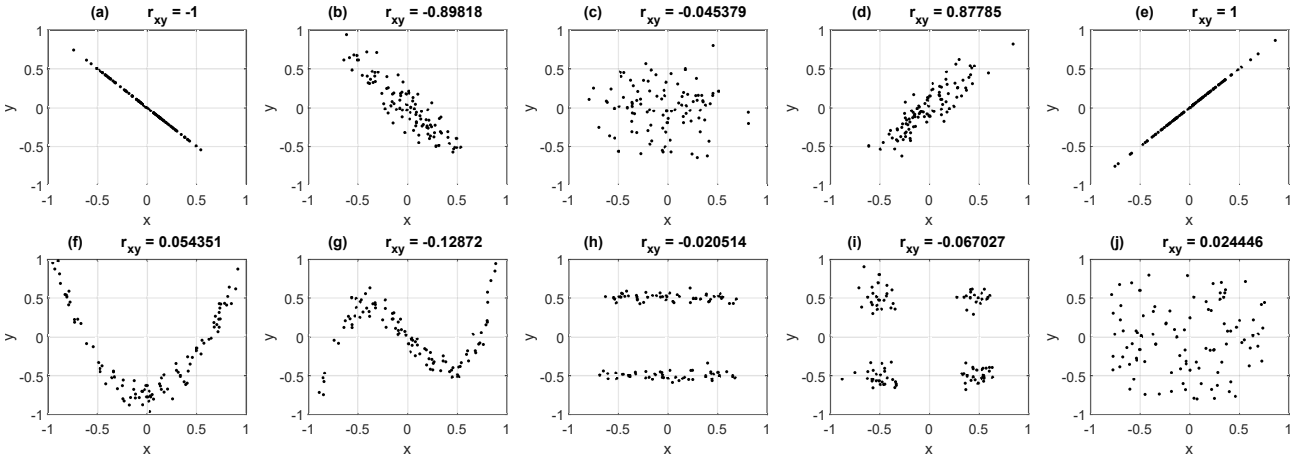


FIGURE 2.3 – Exemples de données de degrés de corrélation variables et coefficients de corrélation associés.

Comme le montre les Figures 2.3 (a) à (e), certaines valeurs caractéristiques de ce coefficients sont très utiles, mais il est important de les considérer avec précaution :

— $r_{xy} = -1$: les séries x et y sont exactement corrélées négativement. On peut calculer y en fonction de x de façon exacte par une relation linéaire.
— $-1 < r_{xy} \leq -0.8$: les données sont fortement corrélées négativement : on peut estimer l'une en fonction de l'autre de manière fiable par une relation linéaire.
— $-0.8 \leq r_{xy} \leq -0.5$: les données sont faiblement corrélées négativement : on peut estimer l'une en fonction de l'autre par une relation linéaire mais avec une forte incertitude.
— $-0.5 \leq r_{xy} \leq +0.5$: les données ne sont pas corrélées linéairement, elles sont probablement indépendantes l'une de l'autre.
— $+0.5 \leq r_{xy} \leq +0.8$: les données sont faiblement corrélées négativement : on peut estimer l'une en fonction de l'autre par une relation linéaire mais avec une forte incertitude.
— $+0.8 \leq r_{xy} < +1$: les données sont fortement corrélées positivement : on peut estimer l'une en fonction de l'autre par une relation linéaire de manière fiable.
— $r_{xy} = +1$: les séries x et y sont exactement corrélées positivement. On peut calculer y en fonction de x de façon exacte par une relation linéaire.

Toute cette analyse des valeurs du coefficient de corrélation n'est valable que pour le recherche de corrélations linéaires. Comme le montrent les Figures 2.3 (f) à (i), ce coefficient ne permet pas de détecter des corrélations plus complexes :

— les Figures 2.3 (f) et (g) présentent des données dont les coordonnées x et y sont corrélées par des polynômes de degrés 2 et 3 respectivement : le coefficient de corrélation est proche de 0. Cela demontre juste qu'il n'existe pas de corrélation linéaire entre x et y, mais ce coefficient ne détecte pas l'existence des corrélations non-linéaires entre les données.

— la présence de clusters au sein des données peut également masquer la détection de corrélations linéaires : sur la Figure 2.3 (h), il est clair qu'il existe deux clusters de données pour lesquels les données sont linéairement corrélées. Pourtant, le coefficient de corrélation ne les détecte pas.

Pour un tableau de données $X = \{x_j^{(i)}\}$ (n lignes correspondant aux points de données, p colonnes correspondant aux variables x_j), on peut calculer des coefficients de corrélation entre les colonnes deux à deux ($r_{j_1 j_2}$ entre les colonnes x_{j_1} et x_{j_2}) et construire une matrice de corrélation R ($p \times p$) :

$$X = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & \cdots & x_p^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & \cdots & x_p^{(2)} \\ x_1^{(3)} & x_2^{(3)} & x_3^{(3)} & \cdots & x_p^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_1^{(n)} & x_2^{(n)} & x_3^{(n)} & \cdots & x_p^{(n)} \end{pmatrix} \rightarrow \boldsymbol{R} = \begin{pmatrix} 1 & r_{12} & r_{13} & \cdots & r_{1p} \\ r_{21} & 1 & r_{23} & \cdots & r_{2p} \\ r_{31} & r_{32} & 1 & \cdots & r_{3p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{p1} & r_{p2} & r_{p3} & \cdots & 1 \end{pmatrix} \tag{2.8}$$

La diagonale de cette matrice de corrélation contient nécessairement des +1 car chaque variable est exactement corrélée positivement à elle-même. La matrice est symétrique car $r_{xy} = r_{yx}$. Les valeurs hors-diagonale permettent de détecter l'existence de corrélations linéaires, donc de réduire éventuellement le nombre de variables nécessaires pour représenter les données : si r_{43} est égal à +1, alors la colonne x_3 ou la colonne x_4 de X peut être supprimée car x_4 et x_3 sont parfaitement corrélées.

2.2 Régression linéaire multiple

La régression linéaire multiple est une extension de la régression linéaire classique, qui permet d'identifier les coefficients de la droite de régression qui passe au mieux par un ensemble de données, au sens des moindres carrés. La généralisation multilinéaire permet de régresser sur ces données un modèle $f()$ dont l'expression est linéaire par rapport aux coefficients inconnus a_j, mais qui peut être non-linéaire par rapport aux variables (x pour un problème monovariable $y = f(x)$, ou x_1, x_2, ..., x_k pour un problème multivariable $y = f(x_1, x_2, \ldots, x_k)$.

Si on note x l'ensemble des variables $x_1, x_2, \ldots, x_k$ et a le vecteur colonne des coefficients à identifier, alors le modèle peut s'écrire :

$$y_{mod} = f(x; a) \tag{2.9}$$

Comme ce modèle est linéaire par rapport aux coefficients inconnus a, on peut aussi l'écrire sous forme d'un produit scalaire, pour une expérience i caractérisée par des variables $x^{(i)}$:

$$y_{mod}^{(i)} = f(x^{(i)}; a) = \sum_{j=1}^{m} a_j \phi_j(x^{(i)}) = \Phi(x^{(i)}).a \tag{2.10}$$

où $\Phi(x^{(i)})$ désigne le "régresseur", c'est-à-dire le vecteur des termes mathématiques, $\phi_j(x^{(i)})$, qui constituent les différents termes du modèle.

Par exemple, pour un modèle polynomial de degré 4 d'un seule variable x, l'expression mathématique du modèle et son régresseur sont :

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 \quad \text{et} \quad \Phi(x) = [1 \quad x \quad x^2 \quad x^3 \quad x^4] \tag{2.11}$$

De même, pour un modèle incluant un terme bilinéaire et des termes quadratiques dépendant de deux variables x_1 et x_2, l'expression mathématique du modèle et du régresseur sont respectivement :

$$y = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_1 x_2 + a_4 x_1^2 + a_5 x_2^2 \quad \text{et} \quad \Phi(x) = [1 \quad x_1 \quad x_2 \quad x_1 x_2 \quad x_1^2 \quad x_2^2] \tag{2.12}$$

Pour une expérience, le régresseur est un vecteur ligne. Pour un ensemble de n expériences, il prend la forme d'une matrice dont chaque ligne i correspond au régresseur de l'expérience i :

$$\Phi = \begin{pmatrix} \phi_1(x^{(1)}) & \phi_2(x^{(1)}) & \cdots & \phi_m(x^{(1)}) \\ \phi_1(x^{(2)}) & \phi_2(x^{(2)}) & \cdots & \phi_m(x^{(2)}) \\ \vdots & \vdots & & \vdots \\ \phi_1(x^{(n)}) & \phi_2(x^{(n)}) & \cdots & \phi_m(x^{(n)}) \end{pmatrix} \tag{2.13}$$

Les coefficients a du modèle peuvent alors être estimés tels que :

$$\hat{a} = (\Phi^T \Phi)^{-1} \Phi^T y \tag{2.14}$$

où y est le vecteur des étiquettes expérimentales. La matrice $(\Phi^T \Phi)$ est appelée matrice d'information et son inverse $(\Phi^T \Phi)^{-1}$ est la matrice de dispersion. La matrice de dispersion est la matrice de covariance des coefficients a_i et a_j du modèle. Sa diagonale renseigne donc sur la variance associée à chaque coefficient a_i du modèle et ses coefficients hors diagonale renseignent sur les covariances entre coefficients, donc sur de possibles couplages. L'optimisation de cette matrice de dispersion est la base des méthodes de planification optimale d'expériences.

Cette matrice de dispersion permet alors d'estimer des intervalles de confiance sur les paramètres a_i du modèle, au niveau de signification α :

$$\hat{a}_i - t_{\alpha/2,(n-m)} s_{a_i} < a_i < \hat{a}_i + t_{\alpha/2,(n-m)} s_{a_i} \tag{2.15}$$

où s_{a_i} est l'écart-type d'estimation du coefficient a_i :

$$\hat{\sigma}^2(a_i) = s_{a_i}^2 = c_{ii} s_y^2 \tag{2.16}$$

où c_{ii} désigne l'élément diagonal de la matrice de dispersion $(\Phi^T \Phi)^{-1}$ et s_y^2 est une estimation de la variance de la régression, aussi appelée variance des résidus :

$$s_y^2 = \frac{1}{n-m} (y - \Phi \hat{a})^T (y - \Phi \hat{a}) \tag{2.17}$$

La variable de Student $t_{\alpha/2,(n-m)}$ qui apparait dans les bornes des intervalles de confiance peut être lue dans les tables dédiées comme le Tableau 2.1.

2.3 Qualité d'un modèle, d'une régression

Plusieurs grandeurs peuvent être calculées ou tracées pour caractériser la qualité d'un modèle ou d'une régression. Dans cette partie, on notera y_i^{exp} l'étiquette "expérimentale" de la mesure i de variables $x_j^{(i)} (1 \leq j \leq p)$, et y_i^{mod} l'estimation de cette étiquette par un modèle ou une régression. Les diagrammes de parité et le coefficient de détermination peuvent être utilisés que le modèle soit linéaire ou non.

2.3.1 Diagrammes de parité

Les diagrammes de parité sont des graphes où on trace généralement y_i^{mod} en fonction de y_i^{exp} pour l'ensemble des données disponibles. La Figure 2.4 présente des exemples de diagrammes de parité de divers modèles. Chaque diagramme représente les données (ronds), la première bissectrice en trait continu et les droites d'écarts à ±10% et ±20% (pointillés) :

- la figure de gauche présente le diagramme de parité d'un très bon modèle : tous les points s'alignent sur la première bissectrice, ce qui indique que la valeur modélisée y_i^{mod} est très proche de l'étiquette y_i^{exp}.
- la figure du centre présente un modèle de qualité moyenne. Les points se répartissent globalement autour de la première bissectrice indiquant que le modèle reproduit bien la tendance générale des mesures, mais il existe tout de même des écarts parfois importants entre y_i^{mod} et y_i^{exp}.
- la figure de droite présente les résultats d'un modèle biaisé : les points ne s'alignent pas du tout avec la bissectrice. Il existe un écart important (biais) qui dépend nettement des valeurs expérimentales.

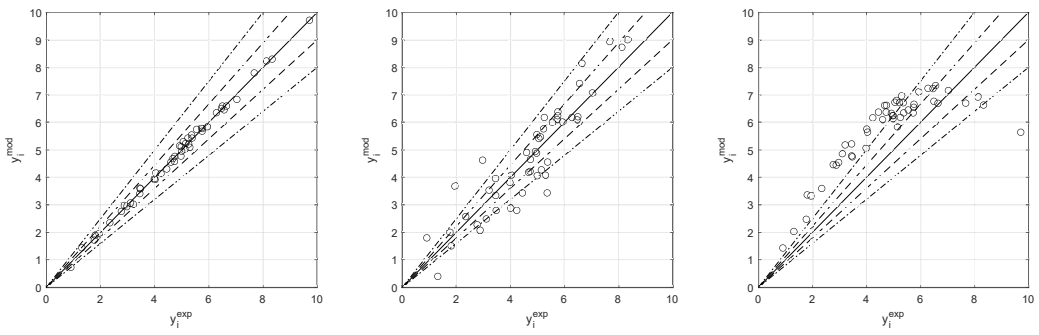


FIGURE 2.4 – Exemples de diagrammes de parité de divers modèles.

ν \ α	0,40	0,30	0,20	0,10	0,050	0,025	0,010	0,005	0,001	0,0005
1	0,325	0,727	1,376	3,078	6,314	12,71	31,82	63,66	318,3	636,6
2	0,289	0,617	1,061	1,886	2,920	4,303	6,965	9,925	22,33	31,60
3	0,277	0,584	0,978	1,638	2,353	3,182	4,541	5,841	10,22	12,94
4	0,271	0,569	0,941	1,533	2,132	2,776	3,747	4,604	7,173	8,610
5	0,267	0,559	0,920	1,476	2,015	2,571	3,365	5,032	5,893	6,859
6	0,265	0,553	0,906	1,440	1,943	2,447	3,143	3,707	5,208	5,959
7	0,263	0,549	0,896	1,415	1,895	2,365	2,998	3,499	4,785	5,405
8	0,262	0,546	0,889	1,397	1,860	2,306	2,896	3,355	4,501	5,041
9	0,261	0,543	0,883	1,383	1,833	2,262	2,821	3,250	4,297	4,781
10	0,260	0,542	0,879	1,372	1,812	2,228	2,764	3,169	4,144	4,587
11	0,260	0,540	0,876	1,363	1,796	2,201	2,718	3,106	4,025	4,437
12	0,259	0,539	0,873	1,356	1,782	2,179	2,681	3,055	3,930	4,318
13	0,259	0,538	0,870	1,350	1,771	2,160	2,650	3,012	3,852	4,221
14	0,258	0,537	0,868	1,345	1,761	2,145	2,624	2,977	3,787	4,140
15	0,258	0,536	0,866	1,341	1,753	2,131	2,602	2,947	3,733	4,073
16	0,258	0,535	0,865	1,337	1,746	2,120	2,583	2,921	3,686	4,015
17	0,257	0,534	0,863	1,333	1,740	2,110	2,567	2,898	3,646	3,965
18	0,257	0,534	0,862	1,330	1,734	2,101	2,552	2,878	3,611	3,922
19	0,257	0,533	0,861	1,328	1,729	2,093	2,539	2,861	3,579	3,883
20	0,257	0,533	0,860	1,325	1,725	2,086	2,528	2,845	3,552	3,850
21	0,257	0,532	0,859	1,323	1,721	2,080	2,518	2,831	3,527	3,819
22	0,256	0,532	0,858	1,321	1,717	2,074	2,508	2,819	3,505	3,792
23	0,256	0,532	0,858	1,319	1,714	2,069	2,500	2,807	3,485	3,767
24	0,256	0,531	0,857	1,318	1,711	2,064	2,492	2,797	3,467	3,745
25	0,256	0,531	0,856	1,316	1,708	2,060	2,485	2.787	3,450	3,725
26	0,256	0,531	0,856	1,315	1,706	2,056	2,479	2,779	3,435	3,707
27	0,256	0,531	0,855	1,314	1,703	2,052	2,473	2,771	3,421	3,690
28	0,256	0,530	0,855	1,313	1,701	2,048	2,467	2,763	3,408	3,674
29	0,256	0,530	0,854	1,311	1,699	2,045	2,462	2,756	3,396	3,659
30	0,256	0,530	0,854	1,310	1,697	2,042	2,457	2,750	3,385	3,646
40	0,255	0,529	0,851	1,303	1,684	2,021	2,423	2,704	3,307	3,551
50	0,255	0,528	0,849	1,298	1,676	2,009	2,403	2,678	3,262	3,495
60	0,254	0,527	0,848	1,296	1,671	2,000	3,390	2,660	3,232	3,460
80	0,254	0,527	0,846	1,292	1,664	1,990	2,374	2,639	3,195	3,415
100	0,254	0,526	0,845	1,290	1,660	1,984	2,365	2,626	3,174	3,389
200	0,254	0,525	0,843	1,286	1,653	1,972	2,345	2,601	3,131	3,339
500	0,253	0,525	0,842	1,283	1,648	1,965	2,334	2,586	3,106	3,310
∞	0,253	0,524	0,842	1,282	1,645	1,960	2,326	2,576	3,090	3,291

TABLE 2.1 – Valeurs de $t_{\alpha,\nu}$ à ν degrés de liberté, ayant la probabilité α d'être dépassée : Prob $(t > t_{\alpha,\nu}) = \alpha$.

2.3.2 Coefficient de détermination

Le coefficient de détermination, R, correspond au coefficient de corrélation entre les étiquettes y_i^{exp} et les estimations correspondantes par le modèle y_i^{mod} :

$$R = \frac{\sum_{i=1}^{n}(y_i^{exp} - \overline{y^{exp}})(y_i^{mod} - \overline{y^{mod}})}{\sqrt{\sum_{i=1}^{n}(y_i^{exp} - \overline{y^{exp}})^2}\sqrt{\sum_{i=1}^{n}(y_i^{mod} - \overline{y^{mod}})^2}} \tag{2.18}$$

où $\overline{y^{exp}}$ et $\overline{y^{mod}}$ correspondent respectivement aux moyennes des étiquettes et des valeurs estimées par le modèle.

Un coefficients de détermination égal à +1 ou très proche de +1 indique que le modèle réprésente fidèlement les étiquettes expérimentales. Des valeurs en dessous de 0.8, voire négatives, traduisent un modèle de qualité faible, voire médiocre, ou possèdant un biais.

2.4 Compromis biais-variance ou biais-complexité

Le question du compromis biais-variance, aussi appelé compromis biais-complexité, n'est pas spécifique aux méthodes d'apprentissage automatique mais se pose de manière générale à chaque fois qu'un modèle doit être construit sur la base de données, généralement expérimentales. Le choix du modèle doit être réfléchi, en accord avec le principe de parcimonie, pour concilier deux aspects contradictoires d'un modèle :

— le biais du modèle : le modèle ne doit pas être biaisé, c'est-à-dire qu'il doit représenter fidèlement les données utilisées pour le construire et en caler les paramètres. En général, plus on augmente le nombre de paramètres, plus on parvient à "coller" aux données, mais on rend alors l'optimisation de ces paramètres de plus en plus compliquée.

— la complexité du modèle : en même temps qu'on augmente le nombre de paramètres, on augmente la complexité du modèle, par sa forme mathématique ou la variété des effets que doivent représenter ces paramètres. Le modèle semble meilleur par rapport aux données initiales, mais cela peut cacher une incapacité du modèle à représenter de nouvelles données : un modèle trop complexe peut être victime de "sur-apprentissage", il a tellement bien appris les données initiales qu'il les connait par coeur et en a même appris les incertitudes, il est donc incapable de représenter fidèlement d'autres valeurs.

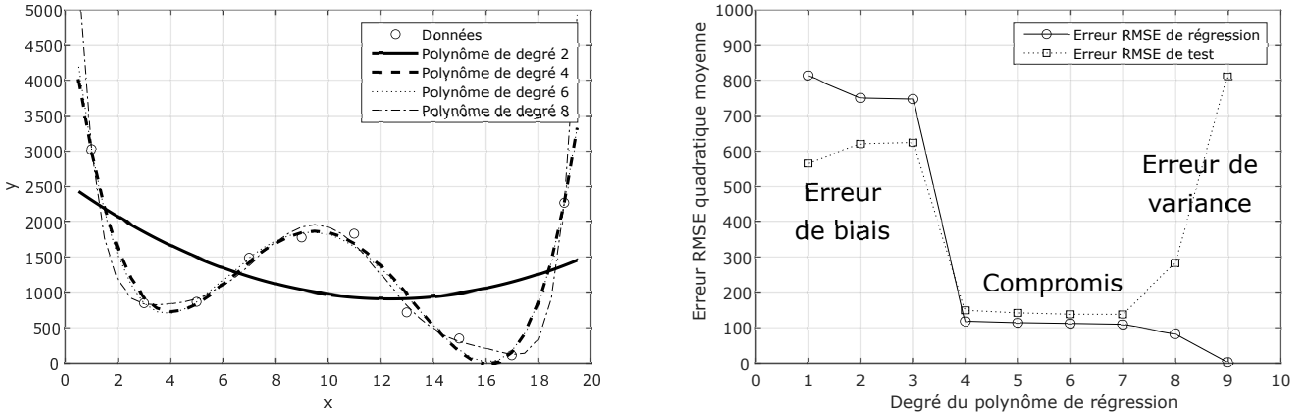


FIGURE 2.5 – Illustration du compromis biais-variance pour la régression polynomiale sur 10 données.

Pour illustrer ces erreurs contradictoires, la Figure 2.5 présente les résultats qu'on peut obtenir lors de la construction d'une régression polynomiale appliquée à 10 mesures expérimentales. La figure de gauche représente les 10 données brutes $y(x)$: on souhaite les modéliser par un polynôme, mais on ne sait pas a priori quel degré utiliser, on va donc faire différents essais en augmentant progressivement le degré du polynôme de 1 à 9. On n'ira pas au-delà de 9, car un polynôme de degré 9 contient 10 paramètres, or nous n'avons que 10 mesures.

Pour chaque degré, on calcule par régression multilinéaire le polynôme de régression de degré d, puis on calcule l'erreur quadratique moyenne RMSE en comparant les mesures expérimentales y_i^{exp} et les mesures estimées par la régression, y_i^{reg}, qu'on appelle erreur d'entraînement :

$$RMSE_{entrainement} = \sqrt{\frac{1}{N_{entrain.}} \sum_{i=1}^{N_{entrain.}} \left(y_i^{reg} - y_i^{exp}\right)^2} \tag{2.19}$$

Ensuite, pour savoir si cette régression de degré d est capable de bien prédire des données qu'elle n'a jamais vu, on utilise 10 autres données, appelées données de test, situées à 10 abscisses aléatoires entre $x = 1$ et $x = 19$, puis on calcule une erreur entre les valeurs de tests et la régression :

$$RMSE_{test} = \sqrt{\frac{1}{N_{test}} \sum_{j=1}^{N_{test}} \left(y_j^{reg} - y_j^{test}\right)^2} \tag{2.20}$$

La partie droite de la Figure 2.5 présente l'évolution de ces deux erreurs, erreur d'entraînement sur les données expérimentales et erreur de test sur de nouvelles données, en fonction du degré du polynôme. La partie gauche de la Figure 2.5 superpose aux données expérimentales les polynômes obtenus aux degrés 2, 4, 6 et 8. L'évolution des erreurs en fonction de d fait apparaître 3 zones :

- $d \leq 3$: pour des degrés faibles, l'erreur d'apprentissage est élevée, de l'ordre de 700, signifiant qu'il y a en moyenne un écart de 700 entre chacun des points expérimentaux et la régression. Comme on le voit sur la figure de gauche, le polynôme de degré 2 ne représente pas correctement les points expérimentaux. Cela semble très naturel car le modèle est beaucoup trop simple pour représenter l'évolution des points : ces modèles possèdent un biais élevé. Puisqu'ils sont mauvais en apprentissage, ces modèles sont aussi mauvais en test et l'erreur de test est du même ordre de grandeur.
- $d \geq 8$: pour des degrés très élevés, l'erreur d'apprentissage est très faible, elle vaut même 0 pour le polynôme de degré 9, ce qui est logique car ce polynôme devient alors un polynôme interpolant et il passe exactement par les 10 mesures expérimentales. Par contre, l'erreur de test devient très grande : ces modèles complexes sont très bons pour reproduire les données utilisées pendant l'apprentissage, mais très mauvais pour prédire des données inconnues. C'est typique d'un comportement de sur-apprentissage.
- $4 \leq d \leq 7$: dès que le degré devient au moins égal à 4, l'erreur d'apprentissage devient très faible avec une erreur moyenne de l'ordre de 100. Sur ce cas particulier, ce comportement est logique car les 10 données d'apprentissage ont préalablement été générées à partir d'un polynôme de degré 4 et bruitées avec un bruit gaussien d'écart-type 200. Sur des données réelles quelconques, la chute n'aurait pas été aussi brutale, mais tout de même visible. Dans cette gamme intermédiaire de degrés, on obtient un compromis entre biais et complexité.

De manière générale, cette démarche est conseillée dès lors qu'on souhaite construire un modèle parcimonieux, c'est-à-dire de complexité suffisante pour capter le comportement des données avec un nombre minimum de paramètres. L'alternance des deux phases de d'apprentissage et de validation se fait sur une ensemble de données qu'on scinde en deux parties : 60 à 70 % des données sont exploitées pendant l'apprentissage et le reste est exploité pour la validation. Une règle heuristique dit que le bon compromis est atteint lorsque l'erreur de validation devient supérieure à celle d'apprentissage ($d = 4$ sur l'exemple) ou lorsqu'elle commence à réaugmenter ($d = 8$ sur l'exemple).

Mathématiquement, on peut démontrer ces effets antagonistes (Himmelblau, 2000 ; Azencott, 2018) par analyse statistique, ce qui mène à la loi suivante :

$$(Erreur)^2 = (Biais)^2 + Variance \tag{2.21}$$

Comme la variance est le carré de l'écart-type, les deux termes de droite sont positifs : pour minimiser l'erreur globale, il faut donc égaliser le "biais" et la "variance". Il n'est pas conseillé de peaufiner un modèle pour le faire coller à la troisième décimale des données expérimentales, il faut être capable d'accepter une certaine erreur de lissage pour obtenir un modèle prédictif avec incertitude.

Chapitre 3

Préparation de données

Toutes les méthodes de traitement de données massives reposant exclusivement sur les données, les résultats qu'elles permettent d'obtenir sont fondamentalement dépendants de la qualité de ces données. Des données de mauvaise qualité fourniront des résultats de mauvaise qualité, que l'on traduit généralement par l'adage :

"Garbage in, Garbage out"

Le prétraitement et la préparation des données sont donc des étapes indispensables, voire primordiales, même si elles ne sont pas les plus passionnantes. Les méthodes de préparation de données visent à :

- **vérifier la complétude et la cohérence des données** : selon le mode d'acquisition (capteur en ligne, enquête d'opinion, etc.), il peut manquer des données ou exister des données aberrantes (panne de réseau, capteur saturé, etc.). Des tracés ou des projections peuvent alors rapidement fournir une information utile.
- **normaliser les données** : les données numériques issues d'un procédé peuvent posséder des ordres de grandeurs extrêmement différents, étalés sur plusieurs puissances de 10. Sans normalisation, selon la méthode appliquée, certaines grandeurs peuvent sembler négligeables et ne pas être considérées par la méthode alors que leur impact sur le procédé est réel et significatif.
- **extraire des descripteurs** : la manipulation de données de grande taille, en plus de rallonger les temps de calcul, peut faire échouer certaines méthodes ou rendre leurs conclusions peu informatives, en diluant l'information qu'on cherche à extraire. Il est souvent préférable de réduire la taille des données élémentaires, en se focalisant sur quelques descripteurs significatifs pour s'assurer que chaque composante apporte une information utile.
- **sélectionner les données pour l'entraînement, le test et la validation** : les méthodes de traitement de données massives possèdent souvent des hyper-paramètres qu'il est difficile de choisir a priori et qu'il faut adapter/optimiser en fonction du problème à résoudre. De plus, ces méthodes sont sujettes à des problèmes de sur-apprentissage ou de sous-apprentissage. Pour toutes ces raisons, il est nécessaire de partitionner les données en données d'entraînement, données de test et données de validation.

3.1 Visualisation

Lorsque les données ont un caractère numérique, comme c'est souvent le cas en Génie des Procédés, il est conseillé de les tracer afin de mieux les visualiser. Cela permet facilement de détecter des incohérences telles que :

- des données manquantes : sur la Figure 3.1(c), il semble qu'une série de données soit manquante entre $t = 55$ s et $t = 77$ s.
- des données mal renseignées : sur la cinquantaine de données de la Figure 3.1(a), alors que la plupart des points ont des valeurs de x_1 et x_2 proches de 10, on observe une dizaine de points qui possèdent un x_1 égal à 0 et une dizaine de points qui ont un x_2 égal à 0. Il est nécessaire de vérifier si ces valeurs nulles sont cohérentes ou issues d'une mauvaise acquisition avant d'appliquer une méthode de traitement.
- des données saturées : la Figure 3.1(b) présente une phase de saturation d'un capteur de pression pour les débits les plus élevés, et la Figure 3.1(c) présente deux phases intermittentes de saturation du signal $y(t)$. L'intégration de ces données dans une méthode de traitement doit être réfléchie car leur présence changera nécessairement le résultat.

Des courbes comme celles de la Figure 3.1 sont possibles en faibles dimensions, mais il devient difficile de visualiser les données dès qu'on dépasse 3 dimensions. On peut néanmoins obtenir des informations sur des corrélations ou des clusters en traçant les données sur des plans de projection ou sous formes de distributions.

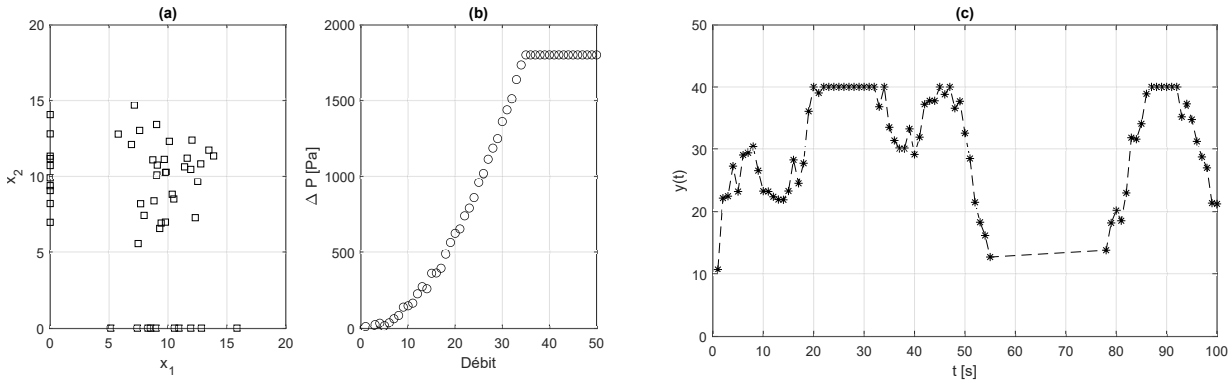


FIGURE 3.1 – Exemples de données possédant des manques ou des saturations.

La Figure 3.2 présente une matrice de projections et distribution d'un ensemble de données dans un espace à 6 dimensions : chaque donnée x est caractérisée par 6 variables x_1, x_2, ... x_6. Sur la diagonale de cette matrice sont tracées les distribution des 6 variables x_i. Au-dessus de la diagonale sont tracées les projection du nuages de données sur les plan $(x_i; x_j)(i \neq j)$: $(x_1; x_2)$, $(x_1; x_3)$, $(x_1; x_4)$, ... $(x_5; x_6)$. Il n'y a rien sous la diagonale car ces figures seraient symétriques à celles au-dessus de la diagonale.

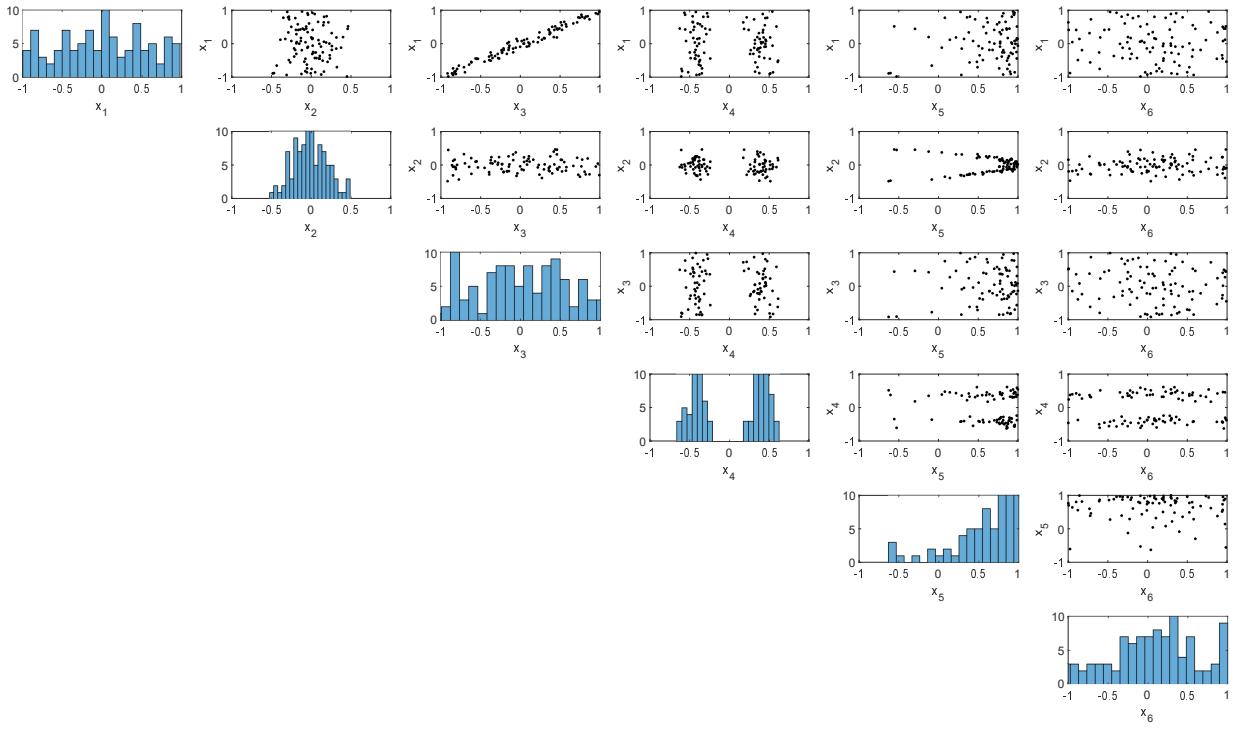


FIGURE 3.2 – Exemple de matrice de projections et de distributions d'un ensemble de données dans un espace à 6 dimensions.

Cette représentation rapide des données peut fournir des informations utiles pour simplifier l'analyse de données ultérieures. Les distributions sur la diagonale présentent des caractéristiques utiles :

— les distributions de x_1 et x_3 ne montrent pas de valeurs préférentielles de ces variables : les données semblent uniformément distribuées sur les intervalles $[-1; +1]$.
— la distribution des valeurs de x_2 est par contre beaucoup plus resserrée autour de 0 selon une distribution qui ressemble à une distribution normale gaussienne.
— la distribution des x_4 est bimodale avec deux groupes de tailles similaires centrées à proximité de -0.5 et +0.5. On peut déjà imaginer qu'il existe deux clusters dans les données, ce qui permettrait de séparer le jeu de données en deux jeux de taille réduite.
— la distribution des x_5 est atypique, fortement asymétrique et majoritairement orientée vers les grandes valeurs de x_5.

Les projections hors-diagonale présentent aussi parfois des formes particulières :

— sur la projection $(x_1; x_3)$, tous les points s'alignent autour de la bissectrice $x_1 = x_3$, ce qui indique une corrélation entre ces variables, même s'il existe un peu de bruit autour de cette corrélation. On pourrait donc réduire le jeu de variables à 5 au lieu de 6 en supprimant x_1 ou x_3, car les informations contenues dans ces deux variables sont redondantes.
— sur la projection $(x_2; x_4)$, on retrouve clairement deux clusters comme l'avait laissé penser la distribution des valeurs de x_4. D'ailleurs, on retrouve cette dichotomie sur tous les graphes faisant apparaître x_4.
— la projection $(x_2; x_5)$ présente aussi une forme particulière, en parabole couchée sur le côté, comme si x_5 pouvait se déduire de x_2 par un polynome de degré 2. L'étude de cette projection mériterait d'être affinée : si une vraie corrélation existe, on pourrait supprimer x_2 car on peut déduire sa valeur en connaissant x_5.

La simple étude des distributions et des projections, sur l'exemple de la Figure 3.2 permet de réduire le problème significativement : d'un problème à n=100 données en d=6 dimensions, on passe à 2 problèmes à 50 données en 4 dimensions. Si une méthode possède une complexité algorithmique en $o(dn^2)$, alors cette simple réduction permet de diviser par 6 le temps de calcul, grâce à la parallélisation des deux problèmes.

Les informations fournies par cette matrice de projections et de distributions sont directement reliées avec la matrice de corrélation présentée en section 2.1.3.

3.2 Normalisation de variables

La normalisation de variables sert généralement à les recentrer sur un intervalle commun de manière à réduire les écarts d'ordres de grandeurs qui pourraient favoriser/défavoriser certaines variables par rapport aux autres.

3.2.1 Normalisation min-max

La normalisation la plus classique consiste à effectuer une changement de variable pour qu'une variable x, varient sur une intervalle $[a; b]$, devienne une variable z contrainte sur l'intervalle $[z_{min}; z_{max}]$. Les équations de transformations s'écrivent alors :

$$z = \frac{z_{min} + z_{max}}{2} + \frac{(z_{max} - z_{min})}{2} \frac{(2x - a - b)}{b - a} \quad \text{et} \quad x = \frac{a + b}{2} + \frac{(b - a)}{2} \frac{(2z - z_{min} - z_{max})}{z_{max} - z_{min}} \tag{3.1}$$

Très souvent, on utilise $z_{min} = -1$ et $z_{max} = +1$, ce qui donne :

$$z = \frac{2x - a - b}{b - a} \quad \text{et} \quad x = \frac{a + b}{2} + \frac{b - a}{2} z \tag{3.2}$$

Pour contraindre z entre 0 et 1, les expressions deviennent :

$$z = \frac{x - a}{b - a} \quad \text{et} \quad x = a + (b - a)z \tag{3.3}$$

La normalisation est très fréquemment appliquée aux variables x_i des données. Elle peut aussi être appliquée aux étiquettes, mais en prenant quelques précautions selon la méthode d'apprentissage qui leur sera appliquée. Par exemple, si les étiquettes sont utilisées en sortie d'un réseau de neurones dont la fonction d'activation de la dernière couche est une sigmoïde (valeur de sortie entre 0 et 1), les étiquettes ne doivent pas être normalisées entre 0 et 1, mais sur un intervalle plus restreint, tel que $[0,1; 0,9]$ afin de conserver une certaine souplesse de représentation.

3.2.2 Variable centrée réduite

Lorsqu'une variable suit une distribution de probabilité particulière, il peut être plus pertinent de normaliser la variable selon des caractéristiques intrinsèques de cette distribution plutôt que dans une gamme min-max.

Par exemple, si une variable x suit une distribution de probabilité normale (gaussienne) de moyenne μ et d'écart-type σ, alors on peut construire la variable z normalisée z telle que :

$$z = \frac{x - \mu}{\sigma} \tag{3.4}$$

La variable z est alors "centrée réduite" : elle suit une loi normale de moyenne égale à 0 et d'écart-type égal à 1. Quelles que soient la valeurs de la variable x, on sait que la variable z possèdera 64 % de ses valeurs entre -1 et +1 et même 95 % de ses valeurs entre -2 et +2. La probabilité que z prenne des valeurs extrêmes, et donc perturbe les méthodes de traitement de données, est quasiment nulle.

3.2.3 Passage au logarithme

Quand une donnée possède une composante qui peut s'étaler sur plusieurs ordres de grandeurs, tel que le débit dans un échangeur tube-calandre (variant de 100 à 10^6 kg/s selon les applications), l'utilisation de la valeur brute donnera probablement plus de poids aux grandes valeurs qu'aux petites. Il est alors préférable de passer de l'échelle linéaire à une échelle logarithmique (décimal ou népérien) :

$$z = \log_{10}(x) \quad \text{et} \quad x = 10^z \tag{3.5}$$

$$\text{ou} \quad z = \ln(x) \quad \text{et} \quad x = e^z \tag{3.6}$$

3.3 Extraction de caractéristiques, de descripteurs, de "features"

Pour les systèmes dont l'espace des variables est de grande dimension et accélérer les méthodes de traitement, il est généralement préférable, pour fiabiliser les résultats de l'analyse, d'utiliser seulement quelques caractéristiques très informatives plutôt que de très nombreuses variables brutes peu informatives. Par exemple, en analyse d'images, des histogrammes de couleurs composés de quelques classes peuvent être plus utiles que des millions de pixels. Ces variables très informatives sont appelées "caractéristiques", "descripteurs" ou "features".

Cette sélection de variables peut être réalisée à différents niveaux en se basant soit sur les variables brutes soit sur de nouvelles variables composites extraites des variables brutes.

Lorsque l'extraction de caractéristiques se base sur les données brutes, des analyses statistiques basées sur la matrice de corrélation et la matrice de projections sont très utiles : elles permettent de supprimer les variables redondantes et de réduire la taille du problème général. Lorsque la dimension des données est restreinte et que les données possèdent un sens physique, un avis d'expert peut aussi être utile pour sélectionner les variables les plus représentatives.

Par exemple, si une méthode doit traiter des données d'équipements tels que des échangeurs de chaleur tube-calandre, l'espace des variables peut être très grand si chaque équipement est caractérisé par ses paramètres géométriques (diamètre de calandre et de tubes, nombre de tubes, nombres de passes, géométrie de chicane, géométries de distributeurs, etc.), ses gammes de conditions opératoires (débits, températures, pressions, etc.), ses matériaux (parois, virole, joints, etc.), ses caractéristiques mécaniques, etc. Selon l'exploitation visée de ces données, ce nombre de variables peut aisément être réduit à quelques grandeurs dimensionnantes : surface d'échange, coefficient d'échage global et gammes de débits.

L'extraction de grandeurs caractéristiques peut aussi être réalisée en construisant de nouvelles variables composites qui permettent de mieux représenter l'ensemble des données : on parle alors de "features" ou de "descripteurs" qui permettent de faire de la réduction de dimensionnalité, dont de travailler dans un espace de variables de plus petite dimension. Ces méthodes seront étudiées plus en détails dans le chapitre 5. On peut rapidement citer l'analyse en composantes principales (ACP), les cartes auto-adaptatives et les réseaux de neurones auto-encodeurs.

3.4 Sélection de données d'entrainement, de test et de validation

Une notion fondamentale associée à la construction de modèles est la notion de généralisation, c'est-à-dire sa capacité à prédire fidèlement des données différentes de celles qui ont été utilisées pour construire ou caler ce modèle. Cette notion est particulièrement importante en apprentissage automatique car le "modèle" n'est pas toujours optimisé de façon déterministe pour représenter les données, mais construit itérativement pour "apprendre" progressivement à mieux représenter les données.

On distingue donc plusieurs types d'erreur qu'un modèle peut effectuer :

— **L'erreur d'entrainement** (fonction de coût) est calculée à partir des données qui servent à entrainer ou à construire le modèle. Pour N données, caractérisées par les variables x_i et les étiquettes y_i, modélisées par un modèle $f()$, l'erreur empirique est la somme des écarts quadratiques entre les étiquettes y_i et leurs modélisations $f(x_i)$:

$$\frac{1}{N}\sum_{i=1}^{N}\big(f(x_i) - y_i\big)^2 \tag{3.7}$$

Cette erreur est généralement celle qu'on cherche à minimiser pour que le modèle représente au mieux les données.

— **L'erreur empirique** est l'erreur calculée sur un jeu de données nouvelles qui n'ont jamais été présentées au modèle pour son entrainement. Son expression est similaire à l'expression précédente où les x_i et y_i sont remplacées par la variables et les étiquettes de ces nouvelles données. Cette erreur empirique est une

estimation de **l'erreur de généralisation** du modèle, donc l'erreur quadratique moyenne que ferait ce modèle sur n'importe quelle autre nouvelle donnée :

$$E\big(\big(f(x)-y\big)^2\big) \tag{3.8}$$

— **L'erreur de modèle** correspond à l'erreur intrinsèque (le biais) qu'un modèle peut occasionner par son incapacité à représenter les données, soit de par sa structure soit de par l'algorithme qui sert à le construire. Cette erreur n'est pas directement quantifiable, elle ne peut être jugée qu'en comparant les erreurs empiriques de différents modèles sur un même lot de données nouvelles. Cette erreur peut aussi résulter des différents réglages des hyper-parapmètres d'une méthode.

Ces différentes sources d'erreur nécessitent des données différentes pour être testées. Dans beaucoup de cas, lorsqu'un ensemble de données est disponible, il est donc nécessaire de partitionner ce lot de données en 3 parties disjointes :

— **Données d'entrainement** : c'est l'ensemble des données utilisées pour construire le modèle par une méthode d'apprentissage automatique. Si plusieurs méthodes ou jeux d'hyper-paramètres doivent être comparés, tous les modèles sont entrainés sur ce même jeu de données, par minimisation de l'erreur d'entrainement.

— **Données de validation** : c'est l'ensemble des données utilisées pour comparer les différentes méthodes ou jeux d'hyper-paramètres. Le modèle fournissant l'erreur empirique minimale sur ces données est alors sélectionné.

— **Données de test** : c'est l'ensemble des données utilisées pour estimer l'erreur de généralisation du modèle retenu lors de l'étape de validation.

Si une seule méthode d'apprentissage est étudiée ou qu'un seul jeu d'hyper-paramètres est retenu, alors l'ensemble des données disponibles peut être partitionné en seulement deux groupes : le jeu de données d'entrainement et le jeu de données de test.

Comme le partitionnement d'un jeu de données en données d'entrainement et de test est une étape arbitraire, on risque de créer des jeux de données non représentatives. Pour éviter ce biais, plusieurs méthodes sont présentées ci-dessous.

3.4.1 Méthode de validation croisée

La méthode de validation croisée consiste à découper le jeu de données D original en K jeux de tailles semblables, $D_k (1 \leq k \leq K)$ (Figure 3.3). Le jeu original est alors itérativement partitionné en deux groupes : les données D_k sont utilisées comme jeu de test (ronds blancs) alors que toutes les autres, $\bigcup_{l \neq k} D_l$, sont utilisées comme données d'entrainement (ronds noirs). Chacune de ces partitions est appélée un *fold*. On définit autant de *folds* qu'il y a de jeux $D_k (1 \leq k \leq K)$.

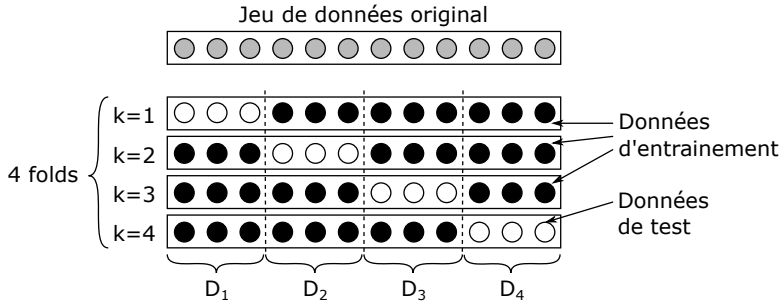


FIGURE 3.3 – Exemple de validation à K *folds* avec $K = 4$.

Chaque *fold* est ensuite utilisé pour entrainer et tester le modèle. Le modèle est donc entrainé et testé K fois. Chaque donnée a été utilisé $K-1$ fois en entrainement et 1 fois en test : une prédiction a donc été obtenue pour chaque donnée originale. On peut alors évaluer la qualité globale du modèle selon deux méthodes :

— Puisque chaque donnée a donné lieu à une prédiction lors de son passage en phase de test, on peut évaluer la qualité de prédiction du modèle sur l'ensemble des données D en utilisant le modèle final.

— On peut également considérer les K modèles intermédiaires obtenus par le traitement de chaque *fold*. La qualité de chaque modèle k est estimée sur ses propres données de test, D_k, et la qualité du modèle global est estimée en moyennant ces qualités partielles.

Un cas extrême de validation croisée est le cas où le nombre de *folds* K est égal au nombre de données N : chaque *fold* ne contient alors qu'une donnée de test, mais $N-1$ données d'entrainement. Cette stratégie

est appelée *leave-one-out* et n'est pas recommandée en raison de son grand temps de calcul et de la difficulté d'interprétation de ses prédiction.

3.4.2 Méthode Bootstrap

Le Bootstrap est une méthode classique en statistiques pour estimer les intervalles de confiance sur les paramètres d'un modèle identifié sur la base de données. Son principe est le suivant :

— Pour un jeu D constitué de N données,
— On constitue B échantillons de données, $D_1, D_2, \ldots, D_b, \ldots, D_B$ en piochant chaque fois N données au hasard, avec remise, parmi les données de D : on obtient donc B échantillons de même taille que le jeu original.
— On entraine B modèles similaires sur ces B échantillons et on teste chacun d'entre eux sur les données de D qui ne sont pas incluses dans son échantillon D_b.
— La comparaison des modèles permet de connaître la sensibilité de leurs paramètres vis-à-vis des données.

Si cette procédure est utile pour estimer les intervalles de confiance de paramètres d'un modèle identifié, grâce à la distribution des valeurs des paramètres des B variantes du modèle, elle est beaucoup moins courante en apprentissage automatique.

3.5 Applications en Génie des Procédés

3.5.1 Matériaux

Goldsmith et al. (2018) présentent une revue généraliste des techniques de machine learning appliquées à la découverte de matériaux pour la catalyse hétérogène, notamment la sélection de descripteurs moléculaires pour représenter ces matériaux.

3.5.2 Détection et diagnostic de fautes et dérives

Himmelblau et al. (1991) ont ajouté un accéléromètre sur une machine tournante pour détecter des anomalies de régimes. L'accéléromètre renvoie un signal, assimilable à un son : ils retraitent ce signal pour extraire des fréquences caractéristiques (descripteurs), qui deviennent les entrées d'un réseau de neurones.

Fan et al. (1993) normalisent leurs mesures entre - π et $+\pi$ pour alimenter un réseau de neurones modifié qui inclut des fonctions sinusoïdales afin d'élargir son potentiel de généralisation grâce à des non-linéarités.

Yoo (2020) développe une méthode améliorée pour la détection d'anomalie en combinant des méthodes de normalisation de variables, de réduction de dimension, de clustering et de recherche de corrélations.

3.5.3 Modélisation d'équipements, d'unités, de procédés

Paoli et al. (2010) appliquent un traitement de données assez complexe à des séries temporelles d'irradiance solaire utilisées pour la caractérisation d'unités photovoltaïques. Les différentes étapes incluent la recherche de données manquantes, la suppression de données aberrantes, la détermination de tendances saisonnières et le recentrage sur ces scénarios saisonniers.

3.5.4 Sélection d'équipements, synthèse et design de procédés

Eilermann et al.(2017) construisent une base de données d'échangeurs (avec un nombre minimal d'équipements) pour couvrir un maximum d'applications industrielles possibles. La première étape consiste à vérifier la complétude des informations, à normaliser les données et à sélectionner des descripteurs cohérents des équipements, compatibles avec les besoins des applications (débits de fluides, coefficients d'échange disponibles, etc.).

Chapitre 4

Clustering, Partitionnement

Les méthodes de partitionnement/*clustering* permettent de réaliser une étude exploiratoire de données non-étiquetées. Elles permettent d'identifier des similarités entre données ayant des caractéristiques communes ou proches, afin de les regrouper ou de définir des classes. Les sous-groupes de données similaires sont appelés clusters. S'il existe des clusters bien distincts, le nombre de données à traiter peut alors être réduit en se focalisant sur l'analyse de données représentatives de chaque cluster plutôt que sur l'ensemble des données.

Le fait de comparer des données entre elles nécessite de pouvoir quantifier un critère de ressemblance ou de dissimilarité. Ce critère prend généralement la forme d'une fonction de distance qui est toujours positive et vaut 0 lorsque deux données sont identiques. Cela ne pose pas de problème si les données sont purement numériques et peuvent être assimilées à des points dans un espace, mais cela peut nécessiter un traitement de données préalables ou la définition d'une fonction de distance.

4.1 Détection de l'existence de clusters

Avant même de lancer une méthode de partitionnement de données, il peut être utile de tester des propriétés globales de ces données pour vérifier, a priori, l'existence ou l'absence de clusters. Une grandeur facile d'accès est la distribution des distances entre paires de données (*pair-wise distance*). Pour un ensemble de N données $x_i (1 \leq i \leq N)$, le calcul de toutes les distances entre paires de données $d(x_i; x_j)(1 \leq i, j \leq N; i \neq j))$ fournit $\frac{N(N-1)}{2}$ valeurs positives qui peuvent être réparties par classes afin de tracer un histogramme de la distribution des distances.

La Figure 4.1 présente 4 exemples de distributions obtenues selon la forme du nuage de données dans un espace à 2 dimensions. Lorsque la distribution des données est uniforme dans l'espace ou qu'elles sont toutes rassemblées en un cluster unique, une distribution en forme de cloche est obtenue. Lorsque les données forment des clusters, la distribution des distances peut faire apparaître des pics, indiquant une structure hétérogène des distances, donc la possible existence de clusters.

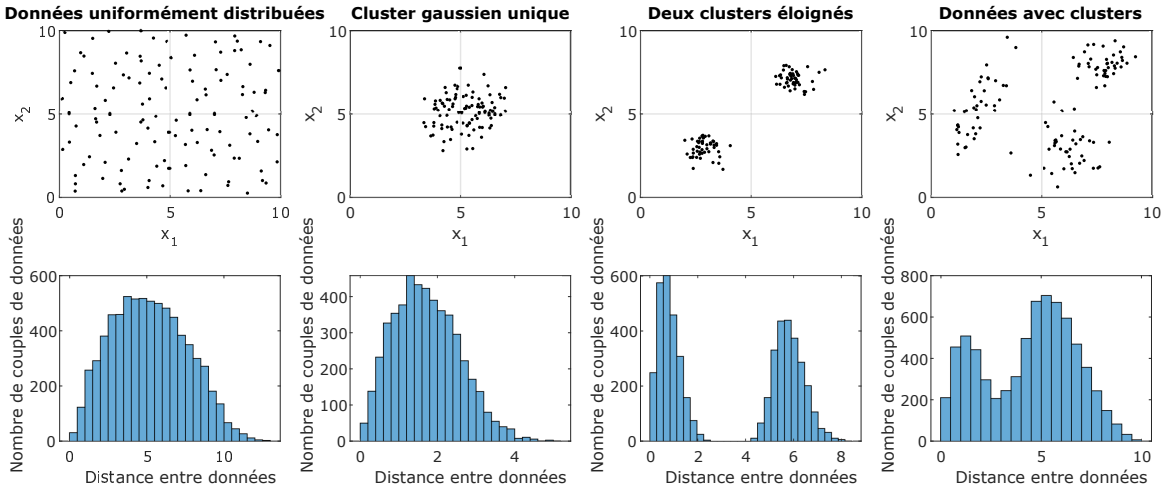


FIGURE 4.1 – Comparaison des distributions de distances entre paires de données pour des données uniformément réparties et des données présentant des clusters.

Cette analyse doit néanmoins être considérée avec précaution et ne remplace en aucun cas une méthode de partitionnement complète. Par exemple, le nombre de pics de la distribution n'est pas égal au nombre de clusters, sauf dans des cas particuliers. De plus, il existe des cas particuliers où les clusters ont une forme allongée qui fait disparaître les pics et donne des distributions en cloches semblables à la répartition uniforme : par exemple, deux clusters constitués de points alignés chacun sur deux droites parallèles fourniront une distribution de distances en cloche.

4.2 Caractéristiques de clusters

4.2.1 Centroïde et médoïde

Pour un ensemble de N_C données x_i $(1 \leq i \leq N_C)$ appartenant à un même cluster C, on définit le centroïde du cluster tel que :

$$\mu_C = \frac{1}{N_C} \sum_{i=1}^{N_C} x_i \tag{4.1}$$

Ce centroïde est le barycentre du cluster, mais n'est pas nécessairement confondu avec une des données qui constituent le cluster. On définit alors le médoïde du cluster comme la donnée la plus proche du centroïde, au sens d'une distance d :

$$m_C = arg_{x_i \in C} \min d\big(x_i, \mu_C\big) \tag{4.2}$$

4.2.2 Homégénéité d'un cluster et d'un partitionnement

Afin de quantifier le degré de ressemblance entre les données d'un unique cluster C_k (contenant N_{C_k} données), on définit l'homogénéité de ce cluster comme la moyenne des distances entre les éléments du cluster et son centroïde :

$$T_k = \frac{1}{N_{C_k}} \sum_{i=1}^{N_{C_k}} d\big(x_i, \mu_{C_k}\big) \tag{4.3}$$

Les distances étant des valeurs positives, l'homogénéité est d'autant plus petite que les données se ressemblent. L'homogénéité vaut 0 lorsque toutes les données sont confondues avec le centroïde.

L'homogénéité globale d'un partitionnement constitué de plusieurs clusters se calcule comme la moyenne des homogénéités des différents clusters.

4.2.3 Séparabilité

La séparabilité de deux clusters C_k et C_l, notée S_{kl} est définie comme la distance entre leurs centroïdes :

$$S_{kl} = d\big(\mu_{C_k}, \mu_{C_l}\big) \tag{4.4}$$

La séparabilité globale d'un partitionnement constitué de K clusters, notée S, se calcule comme la moyenne des séparabilités des clusters deux à deux :

$$S = \frac{2}{K(K-1))} \sum_{k=1}^{K} \sum_{l=k+1}^{K} S_{kl} \tag{4.5}$$

Pour savoir si les données de deux clusters k et l sont susceptibles de se recouvrir, il convient de comparer leurs homogénéités T_k et T_l avec leur séparabilité S_{kl}. Si $\frac{T_k+T_l}{S_{kl}}$ est proche de 0, alors les deux clusters sont très distincts. Une valeur de ce rapport proche de 1 indique une certaine continuité des données sans frontière nette entre les clusters.

L'indice de Davies-Bouldin d'un cluster k au sein d'un partitionnement permet de se faire une idée de sa proximité aux autres clusters :

$$D_k = \max_{l \neq k} \frac{T_k + T_l}{S_{kl}} \tag{4.6}$$

4.2.4 Coefficient de silhouette

Le coefficient de silhouette d'une donnée x_i au sein d'un cluster k, issu d'un partitionnement, permet de savoir si l'affectation de cette donnée à ce cluster est bien justifiée. Pour cela on compare deux distances : d'une part, la distance moyenne entre la donnée x_i et l'ensemble des autres données qui constituent le cluster k, notée $a(x_i)$ et d'autre part la valeur minimale que pourrait prendre $a(x_i)$ si la donnée appartenait à un autre cluster, notée $b(x_i)$:

$$a(x_i) = \frac{1}{N_{C_k} - 1} sum_{x_j \in C_k, j \neq i}\ d\big(x_i, x_j\big) \tag{4.7}$$

$$b(x_i) = \min_{l \neq k} \frac{1}{N_{C_l}} \sum_{u \in C_l} d\big(x_i, u\big) \tag{4.8}$$

Le coefficient de silhouette de la donnée x_i se calcule alors comme :

$$s(x_i) = \frac{b(x_i) - a(x_i)}{\max(a(x_i), b(x_i))} \tag{4.9}$$

La valeur du coefficient de silhouette est d'autant plus proche de 1 que l'affectation de la donnée x_i au cluster C_k est justifiée. Une valeur négative indique qu'il y a recouvrement entre clusters.

4.2.5 Positions et formes des clusters

Une fois qu'une méthode de partitionnement a permis d'aboutir à un ensemble de clusters, ceux-ci peuvent être caractérisés. Les coordonnées de leurs centroïdes renseignent sur leurs positions relatives dans l'espace. Leurs matrices de covariance permettent de savoir s'ils ont une forme plutôt sphérique ou ellipsoïdale : une matrice semblable à l'identité indique une forme sphérique. Si le cluster possède une forme non-sphérique, l'orientation de sa plus grande dimension peut être identifiée grâce au vecteur propre associé à la valeur propre de plus grand module de sa matrice de covariance.

4.3 Méthode des K-moyennes

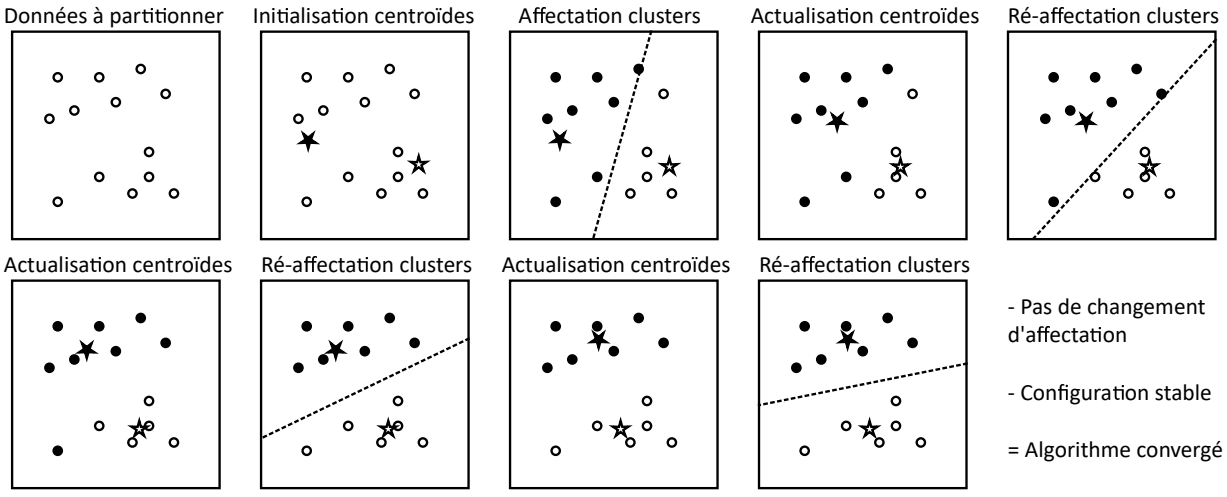


FIGURE 4.2 – Illustration des étapes successives de l'algorithme des K-moyennes.

La méthode des K-moyennes (K-means) permet de partitionner un ensemble de données en K groupes, le nombre de clusters K étant préalablement fixé. Son principe algorithmique est très simple et est visualisé sur la Figure 4.2 :

— On considère un ensemble de données dans un espace où il est possible de calculer une distance entre données et une moyenne sur des valeurs de distance.

— L'algorithme s'applique pour un nombre de clusters potentiels, K, préalablement défini. K est un hyper-paramètre de cet algorithme. Pour l'exemple de la Figure 4.2, on choisit $K = 2$.

— Initialisation des centroïdes : on initialise les coordonnées de K centroïdes, distincts entre eux, dans l'espace des données (représentés par les étoiles blanche et noire). Ces centroïdes peuvent être soit tirés aléatoirement dans cet espace (et donc être distincts des points de données), soit être piochés aléatoirement

parmi les données (et donc être égaux à K données). Cette initialisation est un autre hyper-paramètre de cette méthode.

— Affectation des clusters : en balayant les données, on affecte chacune d'entre elles à un cluster correspondant au centroïde le plus proche d'elle. Ceci est symbolisé sur la figure par une médiatrice en pointillés qui divise ici l'espace en deux zones : toutes les données d'un côté de cette médiatrice sont affectées au cluster du centroïde noir et toutes celles de l'autre côté sont affectées au cluster du centroïde blanc.

— Actualisation des centroïdes : connaissant les données qui constituent chacun des clusters temporaires, on actualise les coordonnées des centroïdes en les remplaçant par les coordonnées des barycentres de chaque cluster. Chaque centroïde se déplace donc vers le centre du nuage de points de son cluster.

— Ré-affectation des clusters : les centroïdes ayant changé, les frontières entre clusters, symbolisées par les médiatrices, ont bougé et certaines données sont donc susceptibles de changer de cluster. Pour chaque donnée, on recalcule donc les distances aux centroïdes et on ré-affecte chaque donnée au cluster du centroïde le plus proche.

— Les opérations d'actualisation des centroïdes et d'affectation des clusters sont répétées itérativement jusqu'à ce que la configuration soit stable, indiquant que l'algorithme a convergé.

Bien que cela n'apparaisse jamais dans l'algorithme, la méthode des K-moyennes permet de minimiser une distance globale grâce au partitionnement, appelée variance intra-cluster. Pour un partitionnement en K clusters, cette variance est la somme, pour chaque cluster $C_k (1 \leq k \leq K)$ des carrés des distances de ses données x_i à son centroïde μ_k, définie telle que :

$$\sum_{k=1}^{K} \sum_{x_i \in C_k} d^2(x_i, \mu_{C_k}) \tag{4.10}$$

Cet algorithme est dit "convergent", c'est-à-dire qu'il convergera toujours vers une configuration stable. Néanmoins, il peut converger vers un optimum local qui ne minimise pas parfaitement la variance intra-cluster, donc vers une partition sous-optimale. Pour augmenter les chances de converger vers une solution globalement optimale, il est conseillé d'exécuter plusieurs fois l'algorithme en changeant l'initialisation.

4.3.1 Choix du nombre de clusters K

L'application de l'algorithme nécessite d'avoir choisi une valeur du nombre de clusters K. Or ce nombre n'est pas connu a priori. Pour déterminer ce nombre de manière fiable, il est nécessaire d'appliquer plusieurs fois la méthode en faisant croître ce nombre K et en suivant l'évolution de la variance intra-cluster en fonction de K.

La Figure 4.3 présente le type de courbe obtenue. Pour chaque valeur de K, les différentes applications de la méthode avec des initialisations différentes ont fourni plusieurs valeurs de la variance intra-cluster, dont la plus petite correspond à la meilleure partition en K clusters. La courbe décrivant l'évolution de ces meilleures variances est décroissante car l'ajout de clusters fait inexorablement diminuer la variance : elle devient même nulle lorsque le nombre de cluster K est égal au nombre de données à partitionner. Cette courbe présente généralement une cassure, un coude, qui indique la valeur de K à retenir. Pour les valeurs de K plus petites, la variance chute très vite, alors que sa décroissance ralentit au-delà. L'absence de coude sur cette courbe peut être un symptôme de l'absence de clusters bien distincts dans les données.

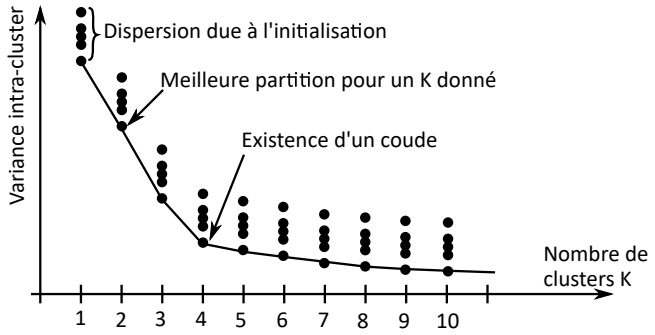


FIGURE 4.3 – Evolution classique de la variance intra-cluster en fonction du nombre de clusters.

4.3.2 Forme des clusters obtenus

Par définition, chaque donnée à partitionner est affectée au cluster dont le centroïde est le plus proche : les frontières entre clusters sont donc des hyperplans médians entre centroïdes. En 2 dimensions, les frontières sont

des segments de médiatrices associées aux couples de clusters. La partition des données issue de la méthode des K-moyennes est donc un diagramme de Voronoï (Figure 4.4). Cela implique que les enveloppes des clusters sont convexes. La méthode des K-moyennes ne peut pas fournir de résultats satisfaisants lorsque les clusters sont imbriqués, entrelassés ou concentriques.

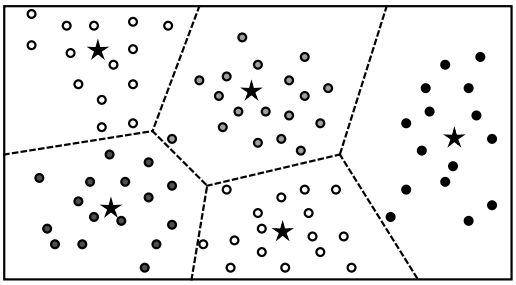


FIGURE 4.4 – Illustration d'une partition par la méthode des K-moyennes faisant apparaître un diagramme de Voronoi.

Il existe plusieurs variantes de la méthode des K-moyennes. La méthode K-means++ réduit les effets d'initialisation en plaçant les centroïdes initiaux de la façon la plus dispersée possible parmi les données. Des variantes exploitant l'astuce du noyau permettent de lever la restriction liée à la convexité des clusters détectés (Azencott, 2019).

4.4 Analyse hiérarchique de clusters

Le clustering hiérarchique consiste à construire séquentiellement les clusters à partir des données sans présupposer de nombre de clusters potentiels, puis à choisir ensuite le nombre de clusters à retenir. On distingue d'une part l'approche agglomérative, qui part des données isolées puis les regroupe progressivement en clusters de taille croissante jusqu'à avoir traité l'ensemble des données, et d'autre part l'approche divisive, qui part d'un cluster unique composé de toutes les données et le divise progressivement en sous-clusters jusqu'à avoir finalement isolé toutes les données. Seule l'approche agglomérative sera présentée ici.

4.4.1 Approche agglomérative et dendrogramme

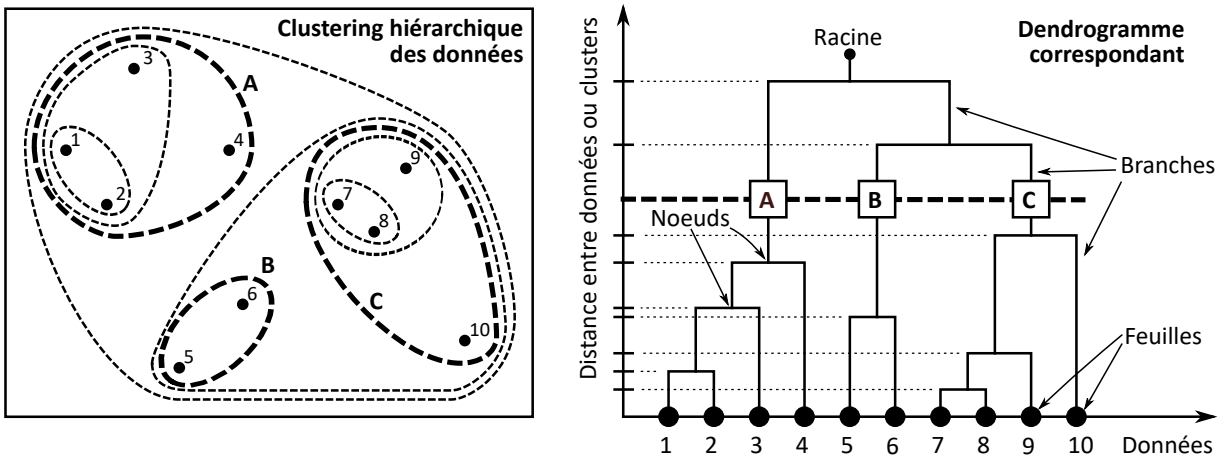


FIGURE 4.5 – Illustration d'une analyse hiérarchique de clusters par approche agglomérative et dendrogramme correspondant.

L'approche agglomérative est illustrée sur la Figure 4.5 : la figure de gauche présente les données en deux dimensions avec les enveloppes des clusters qui vont être construits et la figure de droite présente le dendrogramme issu de la construction.

Le dendrogramme présente la structure hiérarchique des regroupements de données et/ou de clusters qui vont être construits itérativement. Il a la forme d'un arbre inversé : sa "racine" correspond au cluster unique regroupant toutes les données et ses feuilles correspondent aux données isolées chacune dans son propre cluster.

Les échelles intermédiaires représentent les différents clusters et sont constituées de branches et de noeuds. Les noeuds correspondent au fait de fusionner deux clusters en un seul cluster de plus grande taille : l'ordonnée verticale d'un noeud indique la distance qui séparait les deux clusters qu'il fusionne.

Dans le cas des données de la Figure 4.5, la première étape a consisté à calculer toutes les distances entre données : les deux données les plus proches étaient les points 7 et 8 qui ont donc été regroupées en un cluster. L'autre couple le plus proche était ensuite les points 1 et 2 qu'on a à nouveau fusionné : le noeud du cluster (1 ; 2) est plus haut que celui du cluster (7 ; 8) car les points étaient légèrement plus éloignés. A l'étape suivante, la distance minimale reliait le point 9 au cluster (7 ; 8) qui ont donc été fusionné. Ces opérations de fusion et de calcul de distance sont répétées jusqu'à ce que toutes les données aient été rassemblées dans un cluster unique.

Une fois que le dendrogramme a été construit depuis les feuilles vers les racines, il est possible d'identifier le nombre préférentiel de clusters à retenir. Pour cela, on s'intéresse aux différentes hauteurs des noeuds reportées sur l'axe de distance à gauche : les meilleures partitions s'obtiennent en coupant l'arbre à l'endroit où l'écart entre noeuds est le plus grand. Dans le cas de la figure, la meilleure partition semble être obtenue avec 3 clusters, notés A, B et C. En redescendant vers les feuilles depuis la position de coupure, on identifie les trois clusters : A=(1 ; 2 ; 3 ; 4), B=(5 ; 6) et C=(7 ; 8 ; 9 ; 10), dont les enveloppes apparaissent en pointillés épais sur la figure de gauche.

4.4.2 Fonctions de liens

Dans toute la description précédente, la notion de distance n'a pas été explicitée alors qu'elle est centrale pour la construction du dendrogramme. En effet, il existe plusieurs façons de définir une distance entre clusters, selon qu'on s'intéresse, par exemple, à la distance minimale ou maximale entre les données qui constituent les clusters. Puisque ces relations de distance ne sont pas des distances mathématiques à proprement parler, on parle plutôt de "fonctions de liens". Pour la présentation des fonctions de liens, on s'intéressera à deux clusters quelconques C_I et C_J contenant chacun N_I et N_J données, notées x_i et x_j ($1 \leq i \leq N_I$, $1 \leq j \leq N_J$).

Le "lien simple" entre deux clusters est défini comme la distance minimale qui sépare leurs données respectives :

$$d_{simple}(C_I, C_J) = \min_{x_i \in C_I ; x_j \in C_J} d(x_i, x_j) \tag{4.11}$$

Le "lien complet" entre clusters correspond, au contraire, à la distance maximale qui sépare leurs données respectives :

$$d_{complet}(C_I, C_J) = \max_{x_i \in C_I ; x_j \in C_J} d(x_i, x_j) \tag{4.12}$$

On définit aussi le "lien moyen" comme la distance moyenne entre les données des clusters :

$$d_{moyen}(C_I, C_J) = \frac{1}{N_I N_J} \sum_{x_i \in C_I} \sum_{x_j \in C_J} d(x_i, x_j) \tag{4.13}$$

De manière plus naturelle, on peut considérer la distance entre les centroïdes des clusters :

$$d_{centroidal}(C_I, C_J) = d\left(\frac{1}{N_I} \sum_{x_i \in C_I} x_i ; \frac{1}{N_J} \sum_{x_j \in C_J} x_j \right) = d\big(\mu_{C_I} ; \mu_{C_J}\big) \tag{4.14}$$

D'autres critères d'agglomération peuvent être considérés tels que l'homogénéité ou la variance intra-cluster (Azencott, 2019).

4.5 Méthodes basées sur la densité

La méthode des K-moyennes et le clustering hiérarchique présentent des inconvénients qui en limitent parfois la pertinence et la performance. Le premier inconvénient est que ces méthodes assignent, par construction, chaque donnée à un cluster : toutes les données sont classées sans distinction, il n'y a pas de rejet de données aberrantes. Or il se peut que les données contiennent des données aberrantes ou inclassables car isolées : la prise en compte de ces données déforme les clusters naturels et peut conduire à un partitionnement sans intérêt. L'autre inconvénient est lié à la forme des clusters : la méthode des K-moyenne ne peut générer des clusters que selon un diagramme de Voronoï, elle échouera systématiquement avec des clusters entrelacés ou imbriqués.

Les méthodes de partitionnement par densité visent à supprimer ces inconvénients en ne définissant les clusters qu'à partir des points (données) qui se trouvent dans des zones denses. Si un point possède beaucoup de voisins proches, alors il semble naturel qu'il appartienne à un cluster. Par contre, s'il est isolé et qu'aucun autre point n'est proche, il semble inutile de le considérer pour construire un cluster.

4.5.1 Notions de ϵ-voisinage, point intérieur, point frontière et point isolé

Pour une donnée x appartenant à un ensemble de données $D = \{x^{(1)}, x^{(2)}, x^{(3)}, \ldots\}$, on définit son ϵ-voisinage comme le sous-ensemble des données u de D qui se trouvent à une distance de x inférieure à une valeur seuil ϵ préalablement définie :

$$N_\epsilon(x) = \{u \in D; d(x, u) < \epsilon\} \tag{4.15}$$

Le cardinal de ce sous-ensemble fournit donc le nombre de voisins de x qui se trouvent à moins de ϵ de distance : il est utile de remarquer que le point x est aussi comptabilisé parmi les voisins dans cet ϵ-voisinage. La notation ϵ ne signifie pas qu'il s'agit d'une valeur très faible (comme c'est souvent le cas en mathématiques), il s'agit simplement d'une valeur de distance "petite" par comparaison aux distances maximales qui peuvent séparer des éléments de D.

En fonction de ce nombre de voisins, on peut savoir si le point x est isolé ou, au contraire, dans une zone dense. Pour cela, on définit un nouveau seuil, lié à ce nombre de voisins, qu'on note n_{min} (Figure 4.6) :

- $N_\epsilon(x) \geq n_{min}$: si l'ϵ-voisinage de x contient n_{min} points ou plus, alors il se trouve dans une zone dense et donc probablement au coeur d'un cluster : on l'appelle **point intérieur**.
- $1 < N_\epsilon(x) < n_{min}$: si x possède un nombre de voisins inférieur au seuil, il est très proche d'un cluster mais se trouve probablement sur un de ses "bords" : on l'appelle **point frontière**.
- $N_\epsilon(x) = 1$: si x est seul dans son ϵ-voisinage, alors il est isolé. Un point seul ne peut pas être représentatif d'un cluster, par définition même des données massives, donc il s'agit probablement d'un point aberrant : on l'appelle **point aberrant** ("**outlier**").

Les méthodes de clustering par densité exploitent cette classification point intérieur/frontière/outlier pour construire les clusters (Figure 4.6). En effet, on s'attend naturellement à ce que les points qui constituent un cluster soient reliés par des liens de longueur inférieure à ϵ : on peut alors, depuis n'importe quel point d'un cluster, aller vers n'importe quel autre point de ce même cluster en effectuant une série de "sauts" de longueur inférieure à ϵ. C'est la notion de "connexion par densité" (Azencott, 2019).

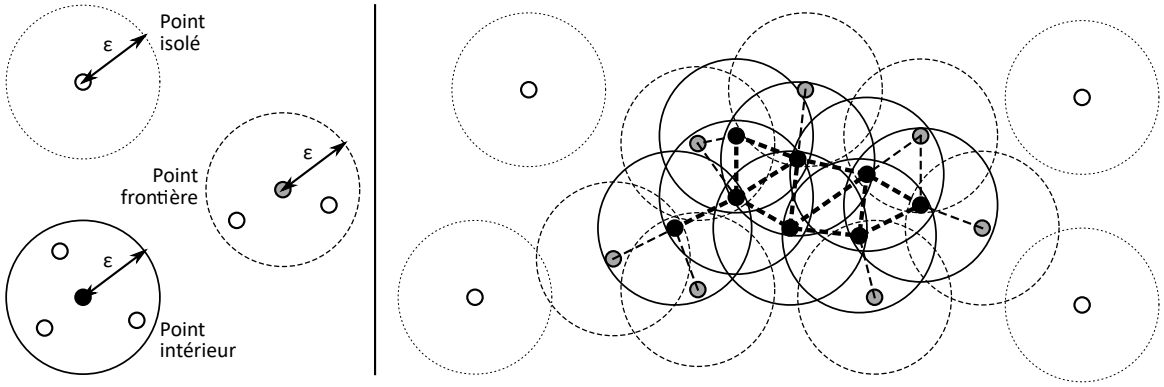


FIGURE 4.6 – Illustration d'un point isolé, d'un point frontière et d'un point intérieur (gauche), et identification des points membres d'un cluster (droite).

4.5.2 Algorithme DBSCAN

L'algorithme DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*) est un algorithme emblématique de partitionnement, proposé en 1996 et primé en 2014 pour sa robustesse et sa pérennité. Il possède 2 hyperparamètres, ϵ et n_{min} (qu'il convient de choisir préalablement et d'optimiser) et fournit la liste des clusters ainsi que la liste des points aberrants.

Son algorithme détaillé est présenté sur la Figure 4.7. Il consiste globalement à initialiser des clusters à partir de "graines" extraites de la liste des points intérieurs, puis à faire croître ces clusters de proche en proche en y incluant les points contenus dans leurs ϵ-voisinages. Lors de cette étape de croissance, si un des voisins appartient déjà à un cluster, on fusionne les deux clusters en un seul.

Malgré ses avantages nombreux (rejet des points aberrants, formes quelconques des clusters, nombre de clusters non-spécifié, efficacité en temps de calcul), la méthode DBSCAN possède deux inconvénients :

- En grande dimension, le "fléau de la dimension" implique que les ϵ-voisinages ont tendance à ne contenir que leur point central : tous les points semblent être isolés.
- Les deux hyperparamètres ϵ et n_{min} figent la densité limite des clusters : la méthode ne permet donc pas d'identifier des clusters de densités différentes.

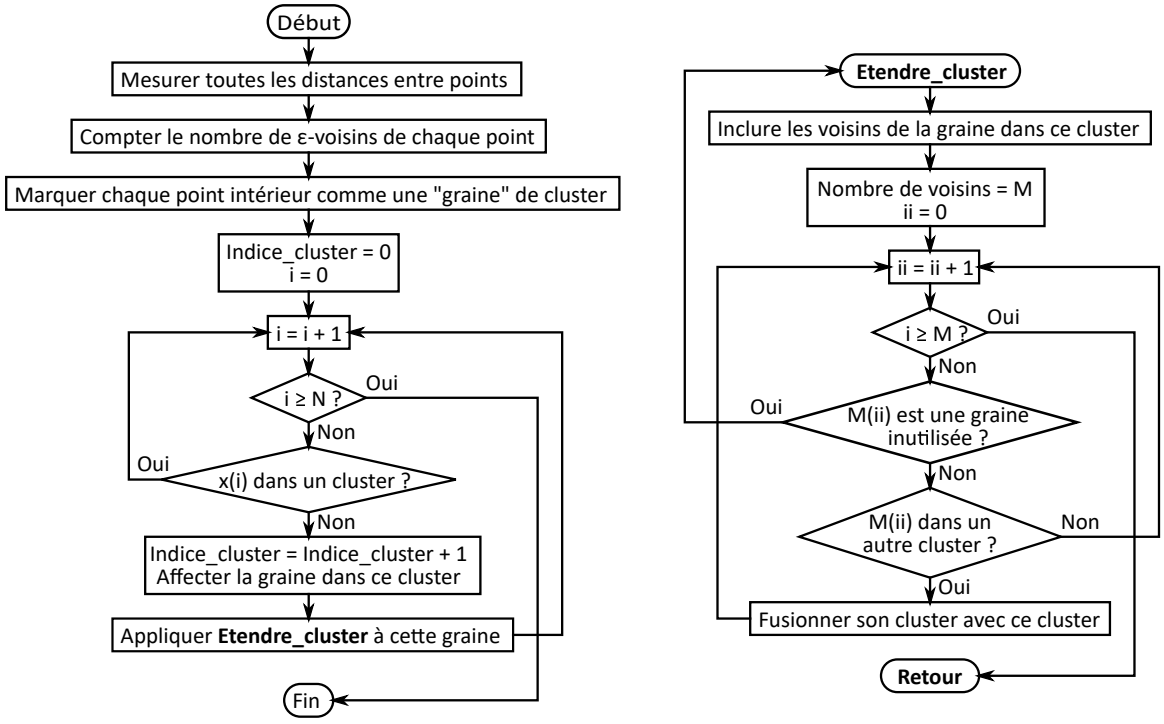


FIGURE 4.7 – Algorithme DBSCAN.

4.6 Propagation d'affinités

La méthode par propagation d'affinité ne nécessite pas de proposer un nombre de clusters à rechercher, mais fournit des clusters linéairement séparables (diagramme de Voronoï). La méthode cherche à regrouper les données autour de données particulières, appelées "exemples", qui sont représentatives de leurs clusters. Initialement, chaque donnée forme un cluster, dont elle est l'unique exemple.

La méthode est basée sur le calcul et l'estimation itérative de plusieurs matrices qui relient les points entre eux :

— La matrice de similarité décrit la ressemblance des données entre elles selon une fonction de distance. Elle est calculée initialement et ne sera plus modifiée.

— La matrice de responsabilité quantifie à quel point une donnée x_j pourrait être un "exemple" de la donnée x_i, par comparaison aux autres données possibles.

— La matrice de disponibilité quantifie à quel point il serait approprié pour x_i de choisir x_j comme "exemple".

Les matrices de responsabilité et de disponibilité sont réactualisées à chaque itération. Les itérations sont stoppées lorsque les clusters sont stables.

4.7 Modèles de mélange gaussiens

Les méthodes de clustering par mélange gaussien visent à identifier la distribution de probabilité de présence des données dans l'espace des variables en assimilant cette distribution à une somme pondérée de distributions gaussiennes multivariables.

Dans un espace à une dimension, la distribution de probabilité gaussienne d'une variable x peut être représentée par une moyenne μ, un écart-type σ et une loi normale (Figure 4.8, gauche) :

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\left(\frac{(x-\mu)^2}{2\sigma^2}\right)\right] \quad , \quad -\infty < x < \infty \tag{4.16}$$

Dans un espace à N dimensions, on peut de manière similaire définir une loi gaussienne à plusieurs dimensions : la Figure 4.8 (droite) représente les courbes iso-densité de la distribution de probabilité dans un espace à 2 dimensions. Cette distribution gaussienne multivariable est caractérisée par :

— une moyenne, qui représente un point particulier dans cet espace que l'on peut assimiler à son centre, là où la densité de probabilité est maximale.

— une matrice de covariance Σ qui informe sur l'étendue et l'orientation de cette distribution de probabilité dans l'espace. Une matrice de covariance semblable à la matrice identité correspond à une distribution sphérique où les surfaces iso-densité sont des sphères concentriques. La présence de valeurs non nulles en dehors de la diagonale indique que les surfaces iso-densité ont la forme d'ellipsoides plus ou moins allongés.

Une telle distribution gaussienne multivariable s'exprime sous la forme :

$$p(x) = \frac{1}{(2\pi)^{N/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right] \tag{4.17}$$

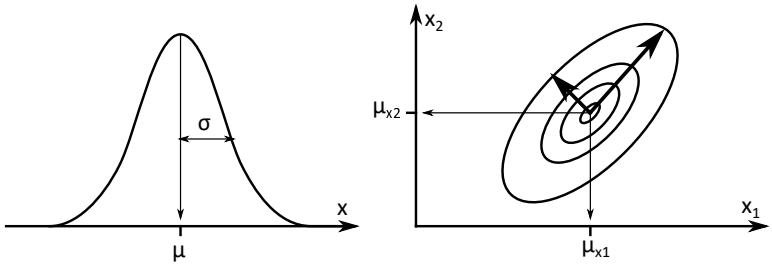


FIGURE 4.8 – Représentation d'une distribution de probabilité normale (gaussienne) monovariable et des courbes iso-valeurs d'une distribution gaussienne multivariable à deux dimensions.

Les méthodes de mélange gaussien cherchent donc à optimiser la position et la forme des ellipsoides des distributions pour "couvrir" au mieux l'ensemble des données à partitionner. La Figure 4.9 présente l'état initial et l'état final de l'application de cette méthode. On initialise la méthode avec un certain nombre de gaussiennes différentes, éventuellement aléatoires : dans l'exemple, on a choisi 2 gaussiennes. Les gaussiennes sont modifiées itérativement en fonction d'un critère de vraisemblance : on teste chaque point pour savoir s'il appartient vraisemblablement à l'une ou l'autre des gaussiennes. Cela permet d'affecter chaque donnée à une gaussienne, comme on affectait chaque donnée à un cluster dans la méthode des k-moyennes. Les caractéristiques μ et Σ des gaussiennes sont alors actualisées. Et ces opérations sont répétées jusqu'à convergence. Il est utile de noter que le nombre de gaussiennes est un hyper-paramètre à optimiser.

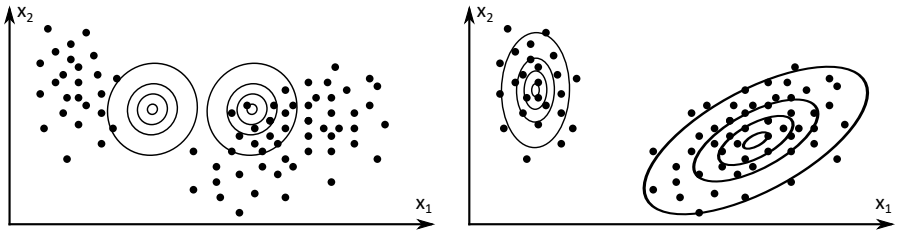


FIGURE 4.9 – Etat initial et état final d'une méthode de partitionnement par modèle de mélange gaussien.

4.8 Mean-shift clustering

La méthode Mean-shift ("Décalage de la moyenne") permet d'identifier, dans une population de données, les "modes" de la distribution des données, c'est-à-dire les maxima de cette distribution, qui correspondent aux zones de plus forte densité. En répétant cette recherche pour toutes les données, la méthode permet de construire les clusters autour de ces modes. Cette méthode ne nécessite donc pas de préciser le nombre de clusters à rechercher et permet de construire des clusters de formes quelconques, mais elle nécessite un grand nombre d'itérations et sa convergence n'est pas assurée.

Dans sa version la plus simple, la procédure de recherche d'un mode à partir d'un point de données contient les étapes suivantes (Figure 4.10) :

— les coordonnées temporaires du mode recherché sont initialisées avec les valeurs du point de départ,
— on définit un voisinage autour de ce point,
— on identifie tous les points présents dans ce voisinage et on calcule leur barycentre,
— on réactualise les coordonnées temporaires du mode recherché avec les coordonnées du barycentre,

— on itère ainsi jusqu'à la stationnarité des coordonnées.

De cette manière, la zone de recherche se déplace dans le sens des densités croissantes et converge vers le maximum local de densité des données. En répétant cette procédure depuis tous les points, on identifie tous les maxima locaux de densité, et on peut donc affecter chaque donnée à un maximum de densité, donc à un cluster. Les clusters correspondent donc aux bassins d'attraction des modes de la distribution de densité.

Les variantes de la méthode se distinguent par le mode de réactualisation des coordonnées, qui peut être soit le barycentre des données dans un voisinage, une fonction calculée sur les K plus proches voisins ou une fonction de distance pondérée calculée sur une population plus large. Le résultat étant dépendant du calcul de moyenne, plusieurs essais doivent être testés pour assurer la validité du résultat. Cette méthode permet d'isoler les points aberrants, qui se retrouvent affectés dans leur propre cluster.

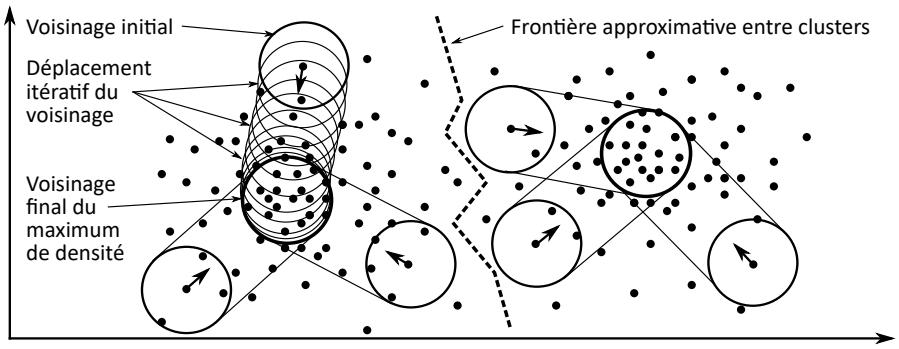


FIGURE 4.10 – Illustration d'une méthode de Mean-Shift Clustering.

4.9 Modèles de Markov

Les méthodes par modèles de Markov sont des méthodes de partitionnement de graphes. Un graphe correspond à un ensemble d'éléments (les noeuds) reliés par des liens (arêtes) éventuellement orientés et pondérés, qui décrivent les interactions entre les noeuds. Dans le cas des modèles de Markov, les noeuds du graphe correspondent aux état d'un système : à chaque instant, le système possède une certaine probabilité d'être dans l'un de ces états.

Les chaines de Markov sont une méthode itérative qui permet de connaître les probabilités que le système soit dans certains états à l'instant t+1 si l'on connait son état à l'instant t. Cette évolution des états probables est calculée sous la forme :

$$v_{t+1}^T = v_t^T * P \tag{4.18}$$

où v_t désigne le vecteur colonne des probabilités des états à l'instant t (taille $\mathrm{N}_{etats} \times 1$), et P la matrice de transition d'un état à un autre (taille $\mathrm{N}_{etats} \times \mathrm{N}_{etats}$). Les valeurs contenues dans la matrice de transition peuvent avoir différentes origines. Il s'agit généralement de raisons physiques liées aux phénomènes qui régissent l'évolution dynamique du système d'un état à un autre. Les clusters regroupent alors des points qui se ressemblent au sens d'une proximité temporelle. Lorsque les états sont en fait des points dans un espace, les valeurs de la matrice de transition sont liées à l'inverse d'une distance spatiale : plus le point i est proche du point j (distance faible), plus la probabilité de passer de i à j est élevée.

La matrice de transition P, ainsi que ses puissances successives, fournit des renseignements sur la structure du graphe. L'existence de blocs renseigne notamment sur l'existence de clusters que forment les états et sur la taille de ces clusters. La Figure 4.11 présente un exemple de réseau et sa matrice de transition : les cases sombres indiquent une valeur élevée de probabilité de transition et les cases claires une probabilité faible. On voit clairement apparaître les deux clusters $\{1\,;\,2\,;\,3\,;\,4\}$ et $\{5\,;\,6\,;\,7\,;\,8\,;\,9\,;\,10\}$ sous formes de deux blocs dans la matrice diagonale par blocs. Les cases grisées hors diagonales traduisent les liens entre les noeuds 2 et 6 d'une part et les noeuds 3 et 8 d'autre part.

4.10 Méthodes spectrales

Les méthodes spectrales s'intéressent également au partitionnement de graphes, représentés par des matrices de liaisons entre les noeuds. Le caractère "spectral" de ces méthodes fait référence au spectre des valeurs propres de ces matrices : $Sp(A) = \{\lambda_i | det(A - \lambda.I) = 0\}$.

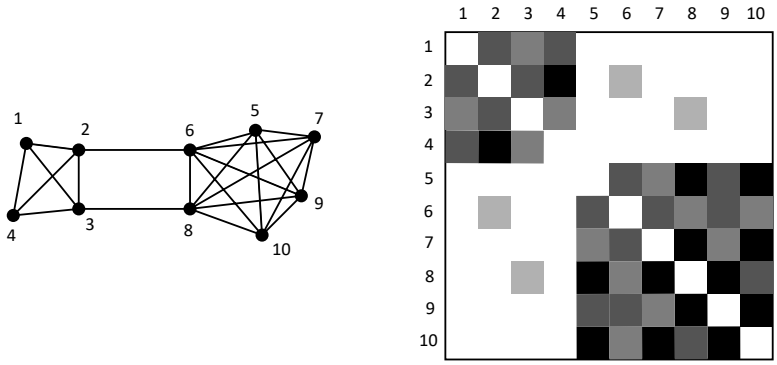


FIGURE 4.11 – Illustration de l'information présente dans la matrice de transition d'un réseau de Markov en vue de son partitionnement.

Un graphe, composé de n noeuds numérotés et m arêtes numérotées, est généralement représenté par 3 matrices :

- La matrice d'incidence, A (taille m × n), composée de 0, -1 et +1 : $A(i,j) = 0$ si l'arête i n'est pas liée au noeud j, $A(i,j) = +1$ si l'arête orientée i arrive au noeud j, $A(i,j) = -1$ si l'arête orientée i part du noeud j,
- La matrice de degré, D (taille n × n), est diagonale et indique le degré de chaque noeud, c'est-à-dire le nombre d'arêtes liées à chaque noeud,
- La matrice d'adjacence, B (taille n × n), contient des 0 partout sauf des 1 lorsque les noeuds i et j sont liés. La diagonale est composée de 0.

A partir de ces matrices élémentaires, on peut construire la matrice laplacienne du graphe, L (taille n × n) :

$$L = A^T.A = D - B \tag{4.19}$$

La matrice Laplacienne du réseau est semi-définie positive et possède une valeur propre égale à 0 associée au vecteur propre $(1;1;1;\ldots;1)$. Les autres vecteurs propres possèdent des composantes positives et négatives : l'étude de ces vecteurs propres permet d'identifier les clusters et de partitionner le graphe. Par exemple, la valeur propre de plus petit module λ_{min} permet de couper le graphe en deux clusters en séparant les composantes positives et négatives de son vecteur propre.

4.11 Cartes auto-adaptatives, Réseaux de Kohonen

Les cartes autoadaptatives sont une technique particulière de réseau de neurones qui permettent de faire du partitionnement, de la classification et de la réduction de dimension. Elles sont présentées dans la section 5.2.

4.12 Applications en Génie des Procédés

4.12.1 Chimie

Chen et Gasteiger (1997) utilisent la méthodes des cartes auto-adaptatives de Kohonen pour catégoriser des réactions de chimie organique (acylations, additions de Michael, etc.) extraites de bases de données.

4.12.2 Matériaux

Corma et al. (2005) appliquent des méthodes de clustering (k-moyennes, réseaux de Kohonen) à des spectres de diffraction de rayons X de matériaux catalytiques pour en extraire des descripteurs.

Fernandez et Barnard (2016) combinent une réduction de dimension par ACP avec un partitionnement par K-moyennes et plusieurs méthodes de classification pour identifier les matériaux MOFs les plus appropriés parmi une banque de 82 000 formulations afin d'optimiser les capacités d'adsorption de CO_2 et N_2.

Spellings et Glotzer (2018) utilisent les modèles de mélange gaussiens pour construire et identifier des clusters à partir de données de structures de cristaux colloïdaux, pour construire des diagrammes de phase.

4.12.3 Détection et diagnostic de fautes et dérives

Yoo (2020) développe une méthode améliorée pour la détection d'anomalie en combinant l'approche classique avec des recherches de corrélations entre variables. La Figure 4.12 illustre lé démarche classique de construction et d'exploitation d'une méthode de détection d'anomalies basée sur des données. Pendant la phase de modélisation, on exploite des données d'entraînement grâce à des méthodes de normalisation, de réduction de dimension et de partitionnement pour modéliser les zones de bon comportement et d'anomalies dans l'espace des variables. Les lois de normalisation, les matrice de passage et les caractéristiques des clusters sont ensuite utilisées, pendant la phase d'exploitation, sur les données réelles du procédé, pour comparer ces nouvelles données aux clusters de bon/mauvais fonctionnement et quantifier un indice d'anomalie permettant de déclencher ou non une alarme. Yoo (2020) ajoute à cette méthode des corrélations empiriques entre données.

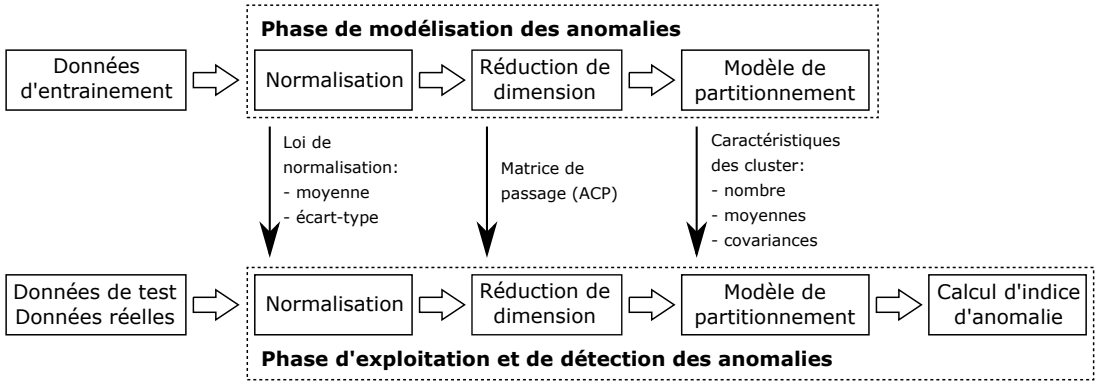


FIGURE 4.12 – Représentation des étapes de construction et d'exploitation d'une méthode de détection d'anomalies basée sur des données.

4.12.4 Sélection d'équipements, synthèse et design de procédés

Eilermann et al. (2017) veulent construire un parc d'échangeurs de chaleur incluant un nombre minimal d'équipements qui pourraient couvrir un maximum de synthèses chimiques possibles. Ils utilisent une méthode de clustering hiérarchique et testent différents nombres de clusters : le représentant de chaque cluster est simulé pour savoir s'il est suffisamment flexible pour assurer différentes applications. Ils parviennent à couvrir 80 % des 275 synthèses visées avec seulement une vingtaine d'échangeurs différents.

Voyant et al. (2017) décrit les méthodes de clustering utilisées pour la prédiction du rayonnement solaire dans les procédés impliquant du solaire photovoltaique ou thermique : K-moyennes, clustering hiérarchique, modèles de mélange Gaussien, etc.

Chapitre 5

Réduction de dimensionnalité

Comme leur nom l'indique, les méthodes de réduction de dimensionnalité visent à réduire la dimensionnalité d'un problème. La dimension correspond ici à la dimension de l'espace de travail, et plus généralement à l'espace des variables. L'idée générale consiste :

— soit à isoler un sous-espace qui contient également toutes les variables,

— soit à construire de nouvelles variables, par une transformation linéaire ou non-linéaire des variables originales, de sorte à ce que le sous-espace formé par cette base de nouvelles variables contienne également toutes les données d'origine. Par exemple, si toutes les données sont coplanaires dans un espace à 3 dimensions, il est plus simple de travailler dans le plan des données que dans l'espace à 3 dimensions complet.

Cette réduction de la dimension du problème permet d'accélérer les méthodes de traitement des données, de fiabiliser leurs résultats en minimisant le fléau de la dimension, et d'identifier de nouveaux descripteurs des données. Les méthodes de réduction de dimension sont des méthodes d'apprentissage non-supervisé car les étiquettes des données ne sont pas considérées.

Ces méthodes doivent néanmoins être appliquées avec prudence car chaque réduction de dimension peut être considérée comme une projection sur un sous-espace. Or une projection crée une perte d'information : deux données qui étaient distinctes, donc différentes, dans l'espace d'origine, peuvent se retrouver superposées, donc identiques, dans le sous-espace de travail. Des caractéristiques importantes peuvent donc être perdues lors de cette étape : les méthodes doivent donc être sélectionnées pour préserver les spécificités des données tout en réduisant la dimension du problème.

5.1 Analyse en Composantes Principales

L'Analyse en Composantes Principales (ACP) est une des méthodes les plus courantes pour réduire la dimensionnalité d'un espace de variables. Elle permet, grâce à une changement de variables linéaire, de trouver une nouvelle base d'expression des variables, telle que la projection des données sur un sous-espace de cette base contient l'essentiel de l'information contenue dans les variables.

La Figure 5.1 illustre cette idée à partir de données dans un espace à 2 dimensions. Initialement, chaque donnée est représentée par ses coordonnées dans le repère $(x_1\,;\,x_2)$: on voit bien ici que les données ne sont pas aléatoires, elles forment un nuage qui possède une dimension étendue selon la première bissectrice (sa "longueur") mais que sa "largeur" est assez restreinte selon la seconde bissectrice. L'ACP va permettre de trouver un nouveau repère $(CP1\,;\,CP2)$ telle que la première composante $CP1$ sera alignée selon la plus grande dimension du nuage de données et la seconde composante $CP2$ selon sa seconde plus grande dimension.

Dans ce nouveau repère, les données s'étalent essentiellement selon la composante $CP1$: on peut facilement les distinguer en regardant cette composante. Par contre, leurs coordonnées selon la composante $CP2$ sont moins étalées et beaucoup de données ont la même composante : cette composante $CP2$ apporte moins d'informations que $CP1$, qui concentre l'essentiel de l'information. Donc, $CP1$ suffit à représenter la diversité des points, on n'a plus besoin de tenir compte de $CP2$: l'ACP a donc permis de passer d'un espace à deux dimensions $(x_1\,;\,x_2)$ à un espace à une seule dimension $CP1$. Néanmoins, l'abandon de $CP1$ a fait disparaître une partie de l'information : la méthode, décrite ci-dessous, vise donc à minimiser les pertes tout en réduisant le nombre total de dimensions.

Pour la suite, on considèrera que les données sont disponibles sous la forme d'un tableau X $(n \times p)$ contenant n mesures dans un espace à p dimensions : chaque mesure i est caractérisée par p variables $(x_{1,i}\,;\,x_{2,i},\,x_{2,i},\,\ldots,\,x_{p,i})$ et constitue une ligne du tableau X.

Les différentes étapes de l'applicaiton de l'ACP sont :

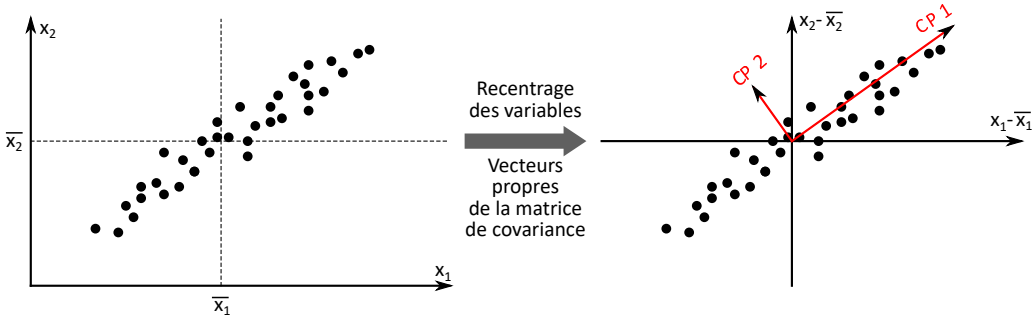


FIGURE 5.1 – Illustration du principe de l'analyse en composantes principales.

— Calcul du point moyen $\bar{X}$:

$$\bar{X} = \left(\sum_{i=1}^{i=n} x_{1,i} \ ; \ \sum_{i=1}^{i=n} x_{2,i} \ ; \ \sum_{i=1}^{i=n} x_{3,i} \ ; \ \dots \ ; \ \sum_{i=1}^{i=n} x_{p,i} \right) \tag{5.1}$$

— Recentrage des données autour du point moyen :

$$B = X - \bar{X} \tag{5.2}$$

— Calcul de la matrice de covariance des données centrée, C :

$$C = B^T.B \tag{5.3}$$

— Calcul des valeurs propres et vecteurs propres de la matrice de covariance C. Comme la matrice de covariance est symétrique et définie positive, les vecteurs propres forment un base orthogonale : il s'agit des axes des composantes principales. La nouvelle base $(CP_1 \, ; \, CP_2 \, ; \, ...)$ est la base des vecteurs propres. De plus, les valeurs propres λ_i de C sont toutes positives et elles sont égales aux variances des données projetées sur les composantes principales. On appelle D la matrice diagonale qui contient les valeurs propres sur sa diagonale (classées par ordre décroissant), et V la matrice des vecteurs propres associés aux valeurs propres de D. Par définition des valeurs propres et vecteurs propres, ces matrices satisfont :

$$C.V = V.D \tag{5.4}$$

— Calcul des coordonnées des points dans la nouvelle base des composantes principales : ces coordonnées s'obtiennent en projetant les points, dont les coordonnées sont dans le tableau B, sur les vecteurs propres de V, par simple produit scalaire :

$$T = B.V \tag{5.5}$$

— Sélection des composantes permettant de réduire la dimension en minimisant la perte d'information. L'information est ici quantifiée par la variance des données projetées sur les composantes principales. Le théorème de partition implique que la varaince totale du nuage de points est égale à la somme des variances des données projetées sur les axes de la base orthogonale des composantes principales. Or ces variances des données projetées sont aussi égales aux valeurs propres de la matrice de covariance. On peut donc calculer la part de variance portée par chaque composante principale :

$$\left(\frac{\lambda_1}{\sum_i \lambda_i} \ ; \ \frac{\lambda_2}{\sum_i \lambda_i} \ ; \ \frac{\lambda_3}{\sum_i \lambda_i} \ ; \ \dots \ ; \ \frac{\lambda_p}{\sum_i \lambda_i} \right) \tag{5.6}$$

— Connaissant la part de variance portée/expliquée par chaque composante, on peut enfin tronquer les dimensions qui n'apportent pas d'informations. La Figure 5.2 présente un exemple de quantification des variances portées par les composantes principales dans un espace à 20 dimensions. Sur le graphe de gauche, on observe que les 3 premières composantes principales représentent chacune entre 17 et 31 % de la variance totale des données. Si on ne garde que ces 3 composantes, on réduite de 20 à 3 le nombre de dimensions, mais on ne conserve que 70 % de la variance totale portée par les données d'origine (graphe de droite). Si cette perte d'information est préjudiciable pour la suite (régression de données, clustering, etc.), alors il faut augmenter le nombre de composantes principales à conserver : il faut garder 7 composantes pour couvrir 90 % de la variance d'origine, et 10 ou 11 si on veut conserver 95 % de la variance. Par contre, ces histogrammes montrent qu'on peut sereinement supprimer 6 dimensions car les 6 dernières composantes n'expliquent aucune partie de la variance totale.

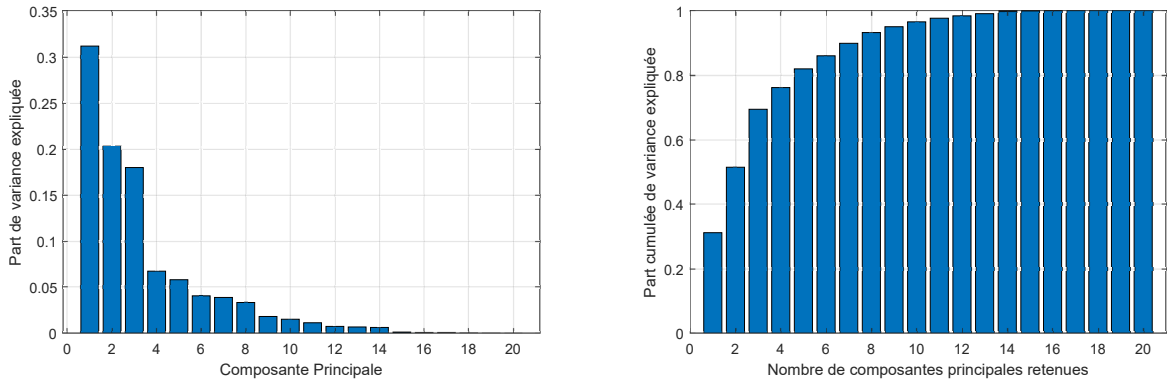


FIGURE 5.2 – Histogramme de la variance expliquée par chaque composante principale (gauche) et de la variance cumulée expliquée par les premières composantes principales (droite).

Cette méthode par Analyse en Composantes Principales est très efficace, mais elle ne propose qu'un changement de variables linéaire : la nouvelle base des composantes est une transformation linéaire de l'espace d'origine. Plusieurs variantes de la méthode permettent d'envisager des effets non-linéaires, parmi lesquelles on peut citer les ICA (Independent Component Analysis) et les méthodes à noyaux KPCA (Kernel Principal Component Analysis) (Yoo, 2020).

5.2 Cartes auto-adaptatives, Réseaux de Kohonen

Les cartes auto-adaptatives sont une forme particulière de réseau de neurones qui permet de cartographier l'espace des données originales sur une carte à 2 dimensions, quelle que soit la dimension des données. Aussi appelées cartes auto-organisées et réseaux de Kohonen, elles permettent de réduire la dimensionnalité d'un problème et de détecter des clusters, même lorsque leurs frontières sont floues et qu'ils se recouvrent. Conceptuellement, la méthode consiste à déformer une surface 2D flexible dans l'espace à p dimensions des données pour que cette surface 2D recouvre le nuage des données quelle que soit sa forme. Le résultat obtenu est présenté sous la forme d'une carte à mailles carrées ou hexagonales (voir Figure 5.3).

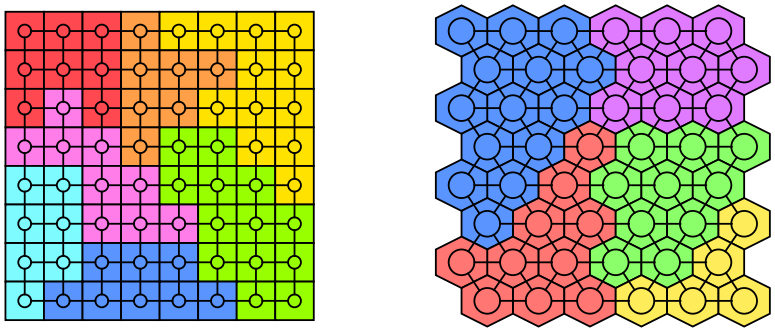


FIGURE 5.3 – Représentation de deux cartes autoadaptatives à mailles carrées (gauche) et à mailles hexagonales (droite).

Chaque case, carrée ou hexagonale, de ces cartes est appelée "neurone" et possède un lien avec les cases voisines : un neurone k est caractérisé par un vecteur de poids $w_j^{(k)}(1 \leq j \leq p)$, de même dimension p que l'espace des données i, notées $x_j^{(i)}(1 \leq j \leq p)$. Chaque neurone peut être assimilé aux coordonnées d'un point dans ce même espace : les neurones et leurs liens forment une sorte de filet. Après entrainement par confrontation aux données, les vecteurs de poids ont été "optimisés" pour que les neurones forment une surface qui "colle" aux données.

Pour comprendre la méthode d'entrainement du réseau et les propriétés finales de la carte, il est utile de le voir comme une matrice à 3 dimensions (Figure 5.4). La carte correspond à sa vue du dessus où chaque case correspond à un neurone. Chaque neurone, donc chaque colonne, a les mêmes dimensions qu'une donnée.

Les étapes d'entrainement sont les suivantes :

— **Initialisation du réseau** : tous les poids $w_j^{(k)}(1 \leq j \leq p)$ des neurones k sont initialisés. L'initialisation

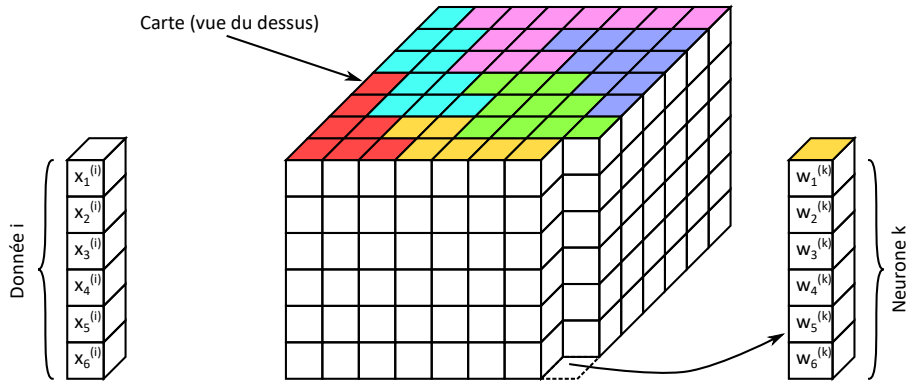


FIGURE 5.4 – Représentation en 3 dimensions du réseau de neurones qui constitue le support d'une carte auto-adaptative, pour des données originalement dans un espace à 6 dimensions.

peut être aléatoire en tirant au hasard chaque $w_j^{(k)}$ dans une gamme compatible avec les valeurs $x_j^{(i)}$ des données. Les neurones peuvent aussi être directement sélectionnés parmi les données.

— **Présentation des données** : les données vont être présentées tour à tour au réseau dans un ordre aléatoire pour le faire évoluer progressivement en actualisant les poids des neurones.

— **Présentation d'une donnée à l'ensemble des neurones** : la donnée i est comparée à chaque neurone k du réseau, permettant de calculer une distance entre cette donnée et ce neurone. Différentes fonctions de distance peuvent être utilisées, mais la distance euclidienne classique est la plus commune :

$$d(i,k) = \sqrt{\sum_{j=1}^{j=p} \left(x_j^{(i)} - w_j^{(k)} \right)^2} \tag{5.7}$$

— **Identification du "neurone gagnant"** : parmi toutes les distances $d(i,k)$ calculées, la plus petite permet d'identifier le "neurone gagnant". Ce neurone possède les poids les plus proches des caractéristiques de la donnée i : c'est le neurone qui ressemble le plus à la donnée i. Dans l'espace des données, le point représentant ce neurone est le point le plus proche de la donnée i parmi tous les points représentant les neurones.

— **Modification du neurone gagnant** : les poids du neurone gagnant sont modifiés de manière à ressembler encore plus à la donnée i. Conceptuellement, cela revient à rapprocher, dans l'espace des variables, le point représentant ce neurone de celui représentant la donnée. De façon simplifiée, mathématiquement, les poids du neurone gagnant d'indice k_{win} deviennent :

$$w_j^{(k_{win},new)} = w_j^{(k_{win})} + \alpha \left(x_j^{(i)} - w_j^{(k_{win})} \right) \quad \text{pour } 1 \leq j \leq p \tag{5.8}$$

où α est un facteur de relaxation ("coefficient d'apprentissage") entre 0 et 1. Si α vaut 1, alors les coordonnées de la donnée $x_j^{(i)}$ sont recopiées à la place des poids du neurones $w_j^{(k)}$. Si α vaut 0, les poids du neurone sont inchangés, ce qui n'aurait aucun intérêt pour entrainer le réseau. En pratique, ce coefficient de relaxation est proche de 1 au début de l'entrainement et décroit progressivement vers 0.

— **Modification des neurones environnants** : l'actualisation des poids ne se limite pas au neurone gagnant, elle est étendue à un voisinage autour du neurone gagnant. Ce voisinage peut être restreint aux premières couches de neurones environnants (Figure 5.5), ou à l'ensemble du réseau via une fonction de distance sur ce réseau : plus le neurone est éloigné, moins ses poids sont impactés par l'actualisation. En pratique, l'actualisation mentionnée ci-dessus s'écrit de manière générale, pour tout neurone k :

$$w_j^{(k,new)} = w_j^{(k)} + \alpha(t)\sigma(t) \left(x_j^{(i)} - w_j^{(k_{win})} \right) \tag{5.9}$$

$$\text{avec } \alpha(t) = \alpha_0 \exp\left(-\frac{t}{t_0} \right) \text{ et } \sigma(t) = \exp\left(-\frac{d_{voisinage}(w^{(k)}, w^{(k_{win})})}{2\sigma_0^2} \right) \tag{5.10}$$

où t désigne l'itération ou l'epoch, $\alpha(t)$ la fonction évolutive de relaxation caractérisée par une valeur initiale α_0 et un temps caractéristique de décroissance t_0, $\sigma(t)$ la fonction de voisinage caractérisée par une fonction de distance entre neurones sur le réseau $d_{voisinage}(w^{(k)}, w^{(k_{win})})$ et un coefficient de voisinage σ_0.

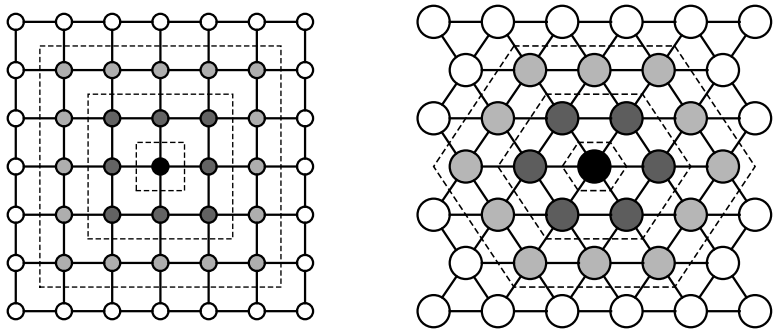


FIGURE 5.5 – Représentation, au sein d'une carte auto-adaptative, d'un neurone gagnant (rond noir central) et des différentes couches de neurones adjacents les plus proches (gris).

— **Itération de l'actualisation** : les étapes précédentes sont répétées jusqu'à stabilisation des poids.

Cet algorithme possède donc plusieurs hyperparamètres : ceux de la fonction de relaxation (α_0 et t_0), ceux de la fonction de voisinage ($d_{voisinage}$ et σ_0), et le nombre d'epochs, c'est-à-dire le nombre de fois que l'ensemble des données sera présenté au réseau pour le faire évoluer.

Après convergence, les neurones ressemblent aux données : tous les points représentant les neurones dans l'espace des données forment un maillage, un filet élastique, qui recouvre le nuage de données. L'actualisation des neurones au voisinage du neurone gagnant implique que des neurones adjacents dans la carte se ressemblent également : ces ressemblances permettent de délimiter des clusters. Cette délimitation est réalisée en calculant, pour chaque lien de la carte, une distance entre deux neurones voisins k_1 et k_2 :

$$d(k_1, k_2) = \sqrt{\sum_{j=1}^{j=p} \left(w_j^{(k_1)} - w_j^{(k_2)} \right)^2} \tag{5.11}$$

Si cette distance est faible, cela confirme que les deux neurones sont proches et font partie d'un même cluster. Par contre, si cette distance est grande, les neurones, bien qu'adjacents sur la carte, sont éloignés dans l'espace des données et appartiennent à des clusters distincts. La visualisation de ces distances de lien entre neurones sur la carte permet de tracer les frontières entre neurones, donc entre clusters. La cartographie des différentes couches horizontales de la matrice 3D du réseau permet ensuite de caractériser chaque cluster en fonction des valeurs de ses composantes de poids w_j, donc des variables x_j des données.

Les cartes auto-adaptatives ont une utilité très large :

— Elles permettent d'identifier l'existence de clusters, de les dénombrer (sans connaître leur nombre a priori) et de les caractériser en termes de variables.

— Elles permettent de classifier de nouvelles données : une nouvelle donnée peut être présentée au réseau après entrainement, on déterminera alors un neurone gagnant et le cluster à laquelle cette donnée peut être rattachée.

— Elles permettent de faire de la réduction de dimensionnalité : comme chaque donnée peut être placée sur un neurone gagnant de la carte, on peut lui affecter des coordonnées en 2 dimensions. On peut même interpoler entre deux neurones pour affiner ces coordonnées : quelle que soit la dimension originale de l'espace des données, chaque donnée peut ainsi être caractérisée par deux nouvelles coordonnées sur cette carte. Le caractère discriminant de ces coordonnées doit être considéré avec précaution car elles correspondent aux coordonnées de projection (au plus proche) du point de la donnée dans un espace à p dimensions sur une surface 2D déformée de façon non-linéaire dans cet espace.

5.3 Réseaux de neurones auto-encodeurs

Un réseau auto-encodeur est un réseau de neurones à propagation avant, donc un perceptron multicouche (voir Chapitre 7). Sa couche d'entrées et sa couche de sorties possèdent exactement le même nombre de neurones car son objectif général est d'être capable de reproduire les données d'entrée (donnée x de variables $(x_1; x_2; \dots x_p)$). Il possède une forme convergent-divergent, généralement symétrique, avec une couche centrale de taille plus petite (Figure 5.6).

Les sorties des neurones de cette couche d'étranglement sont appelées variables latentes. La partie convergente joue le rôle d'encodeur car l'information contenue dans les variables d'entrée (correspondant aux coordonnées d'une donnée dans l'espace des variables) se retrouve comprimée dans les variables latentes. A contrario,

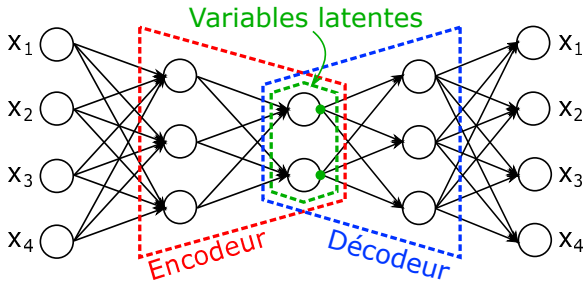


FIGURE 5.6 – Représentation d'un réseau auto-encodeur, avec un espace des variables de dimension 4, une couche cachée d'encodage, une couche latente et une couche cachée de décodage.

la partie divergente du réseau joue le rôle de décodeur car cette partie décompresse l'information des variables latentes pour régénérer les variables originales.

Si l'entrainement du réseau de neurones permet de réduire significativement l'erreur d'entrainement (voir Chapitre 7), cela signifie que l'espace des variables originales peut être réduit à l'espace des variables latentes et qu'il existe une bijection entre ces deux espaces. Le nombre de variables latentes, donc le nombre de neurones dans la couche d'étranglement, peut être choisi pour réduire l'erreur d'entrainement. Un nombre de 2 variables latentes est souvet utilisé pour tracer une carte de l'espace des variables latentes pour détecter visuellement des clusters.

L'encodeur et le décodeur peuvent contenir plus d'une couche cachée. Le nombre de couches est un hyper-paramètres à optimiser. Lorsque le nombre de couches est figé, le nombre de neurones dans chaque couche peut être fixé par une heuristique qui préconise que les nombres de neurones par couche dans l'encodeur et le décodeur varient selon une progression géométrique entre le nombre de neurones de la couche d'entrée et celui de la couche latente. Ainsi, pour des données dans un espace à 10 dimensions ($N_i = 10$) et 2 variables latentes ($N_L = 2$), si on insère une couche cachée d'encodage, il est conseillé d'y inclure $\sqrt{N_i.N_L} = \sqrt{20} = 4.47$, soit 4 ou 5 neurones. Cette heuristique permet de comprimer progressivement l'information contenue dans les variables originales.

Comme les réseaux auto-encodeurs sont des réseaux à propagation avant, ils peuvent être entrainés par les méthodes classiques de rétropropagation (voir Chapitre 7). Pour les réseaux symétriques, même multicouches, il existe aussi une méthode alternative qui consiste à entrainer successivement les différentes couches de l'encodeur et du décodeur depuis les plus grandes vers les plus petites (Figure 5.7).

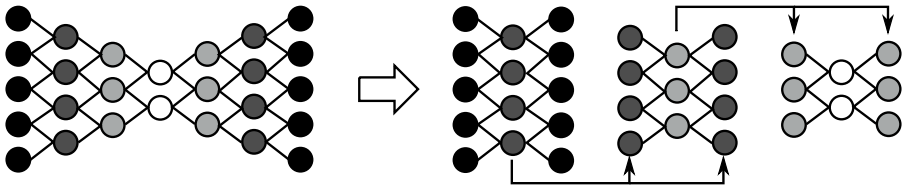


FIGURE 5.7 – Représentation d'un réseau auto-encodeur multicouche (gauche) et d'une méthode récurrente d'entrainement appliquée aux différentes couches (droite).

Les réseaux auto-encodeurs sont une forme non-linéaire d'analyse en composantes principales puisque les variables latentes dépendent non-linéairement des variables d'entrée. Ils ne sont pas seulement utilisés pour de la réduction de la dimensionnalité, ils sont de plus en plus utilisés en modélisation générative. On peut également citer les auto-encodeurs parcimonieux qui incluent un terme de régularisation, et les auto-encodeurs débruiteurs qui sont entrainés à reproduire x à partir de version bruitées de x (Goodfellow et al., 2018)

Les réseaux auto-encodeurs peuvent être décrits comme un empilement de machines de Boltzmann restreintes, qui sont des réseaux de neurones dont les connexions sont à double sens, et qui créent une relation entre une couche visible et une (ou des) couche(s) cachée(s). Ces structures sont utilisées dans des modèles génératifs profonds, qui sortent du cadre de ce cours. Le lecteur intéressé pourra consulter Azencott (2019) et Goodfellow et al. (2018).

5.4 Positionnement multi-dimensionnel

Le positionnement multi-dimensionnel cherche à réduire l'espace des variables tout en conservant une grandeur caractéristique de leurs similarités, et présente beaucoup de points communs avec l'ACP : elle est aussi

appelée "ACP d'un tableau de distances". Le cas le plus simple consiste à définir la similarité des données par rapport à une distance dans l'espace des variables : plus elles sont proches (plus la distance est faible), plus elles sont similaires. Pour deux données distinctes $x^{(i)}$ et $x^{(j)}$ (de taille n, dans $\mathbb{R}^n$) parmi N données disponibles $(1 \leq i, j \leq N)$, on peut définir la distance quadratique euclidienne telle que :

$$d_{ij} = d(x^{(i)}, x^{(j)}) = \sum_{k=1}^{k=n} \left(x_k^{(i)} - x_k^{(j)}\right)^2 \tag{5.12}$$

On obtient alors une matrice de distance D_x (taille $N \times N$) d'élément courant d_{ij}. Comme la matrice de covariance en ACP, cette matrice de distances D est positive et symétrique.

Le positionnement multi-dimensionnel consiste alors à trouver un ensemble de N vecteurs $y^{(1)}$, $y^{(2)}$, ... , $y^{(N)}$ (en nombre égal au nombre de données initiales), de dimension m plus petite que n ($y^{(k)}$ dans $\mathbb{R}^m$, avec $m < n$), tels que le tableau de distances de ces $y^{(k)}$, D_y, soit aussi proche que possible du tableau des distances originales :

$$y^{(k)}(1 \leq k \leq N) = argmin_{y^{(k)}} \sum_{i=1}^{i=N} \sum_{j=1}^{j=N} \Big(D_x(i,j) - D_y(i,j) \Big)^2 \tag{5.13}$$

Le positionnement multi-dimensionnel peut être vu comme une forme de projection sur un sous-espace qui conserve le tableau des distances. Les vecteurs $y^{(k)}(1 \leq k \leq N)$ sont les coordonnées des projections des données $x^{(k)}(1 \leq k \leq N)$ dans ce sous-espace.

En pratique, il est d'abord nécessaire de centrer les données originales $x^{(i)}$ par rapport à leur moyenne $\bar{x} = \sum_{i=1}^{N} x^{(i)}$. On calcule alors les vecteurs propres et valeurs propres de la matrice de distance quadratique des données centrées. Le classement des valeurs propres par ordre décroissant permet de choisir le nombre de dimensions m à retenir. Si on note $\Lambda_m(m \times m)$ la matrice diagonale des m valeurs propres retenues classées, et V_m la matrice des vecteurs propres correspondants en colonnes $(N \times m)$, les vecteurs $y^{(k)}$ recherchés sont alors les colonnes de $Y = \Lambda_m^{1/2} V_m^T$.

5.5 t-SNE

La logique générale de la méthode t-SNE (*t-Student Neighborhood Embedding*) est semblable à l'ACP et au positionnement multi-dimensionnel : on cherche à remplacer les N données originales x incluses dans $\mathbb{R}^n$ par des projections y dans un sous-espace $\mathbb{R}^m (m < n)$ en conservant une propriété des données de départ.

Alors que l'ACP cherche à conserver les covariances et que le positionnement multi-dimensionnel cherche à conserver la matrice de distances, la grandeur que la méthode t-SNE cherche à conserver est une distribution statistique liée à la similarité des données, que l'on peut interpréter comme une probabilité de ressemblance, normalisée par la densité locale des données. Cette distribution peut être calculée sur les données originales x. Elle est notée P.

La méthode considère que cette distribution statistique doit prendre la forme d'une distribution de Student à 1 degré de liberté dans le sous-espace des images y. Elle est notée Q. La méthode consiste alors à optimiser les y afin de minimiser la différence entre P et Q : comme il s'agit de deux distributions, le critère de divergence de Kullback–Leibler est utilisé.

En raison de sa complexité, la méthode ne sera pas détaillée ici. Le lecteur pourra se référer à Azencott (2019) et Sauvalle (2020).

5.6 DPLS - Discriminant Partial Least Squares

La méthode des Moindres carrés partiels (PLS, *Partial Least Squares*, plus justement appelée *Projection to Latent Structure*) est habituellement utilisée dans des problèmes de régression pour la construction de modèles linéaires entre les étiquettes Y et les variables X des données, qui peuvent s'écrire sous le forme

$$Y = B.X + \epsilon \tag{5.14}$$

Néanmois, au lieu de construire directement ce modèle, la méthode passe tout d'abord par une représentation de X et Y dans un sous-espace de variables latentes. Cette étape est particulièrement utile pour construire des modèles où le nombre de paramètres est supérieur au nombre d'observations, ce que ne peut pas faire la régression classique. Ce passage par des variables latentes rend cette méthode compatible avec un objectif de réduction de dimensionnalité.

Sa variante DPLS est une technique de réduction de dimension qui vise à maximiser la covariance entre le prédicteur du bloc des variables X et le prédicteur du bloc des étiquettes Y (Chiang et al., 2000). Comme les étiquettes sont également impliquées, il s'agit d'une méthode de réduction supervisée. La méthode de base est linéaire et n'apporte aucune améliroation par rapport à l'ACP, mais des variantes non-linéaires, et même dynamiques existent.

5.7 Analyse discriminante de Fisher

L'analyse discriminante de Fisher (FDA, *Fisher Discriminant Analysis*) est une méthode de réduction de dimension qui cherche, en plus de la projection dans un sous-espace, à maximiser la séparabilité entre classes de données, c'est-à-dire entre clusters. C'est donc une méthode qui est utilisée en combinaison avec une méthode de clustering ou une méthode de classification.

Si les données x, de moyenne globale $\bar{x}$, peuvent être partitionnées en N_C clusters de centroïdes $\mu_c (1 \leq c \leq N_C)$, on peut calculer les matrices de dispersion inter-clusters, S_{inter}, et intra-clusters, S_{intra} (semblables à un facteur près aux matrices de covariance) :

$$S_{inter} = \sum_{c=1}^{N_C} (\mu_c - \bar{x})(\mu_c - \bar{x})^T \quad \text{et} \quad S_{intra} = \sum_{c=1}^{N_C} \sum_{i \in c} (x_i - \mu_c)(x_i - \mu_c)^T \tag{5.15}$$

Afin de conserver la séparabilité des clusters lors de la projection dans le sous-espace latent, la méthode cherche à maximiser la dispersion inter-clusters par rapport à la dispersion intra-clusters, donc à maximiser le critère $J(w)$, où w est un vecteur de projection :

$$J(w) = \frac{w^T S_{inter} w}{w^T S_{intra} w} \tag{5.16}$$

La maximisation de ce critère d'optimisation fait apparaître un problème intermédiaire de valeurs et vecteurs propres :

$$S_{inter}^{1/2} S_{intra}^{-1} S_{inter}^{1/2} v = \lambda v \tag{5.17}$$

que l'on peut résoudre numériquement pour trouver les vecteurs propres v_k et les valeurs propres λ_k, qui sont toutes positives car la matrice $S_{inter}^{1/2} S_{intra}^{-1} S_{inter}^{1/2}$ est symétrique définie positive

Les vecteurs de projection w_k se calculent alors tels que :

$$w_k = S_{intra}^{-1/2} v_k \tag{5.18}$$

Comme pour l'ACP, les vecteurs de projection à retenir peuvent être choisis en retenant ceux issus des vecteurs propres v_k associés aux plus grandes valeurs propres λ_k.

5.8 Factorisation en matrices non-négatives

La factorisation en matrices non négatives propose une factorisation de la matrice des données X (n données $\times$ p variables) en deux matrices positives de plus petites dimensions, ce qui permet donc une réduction de la dimension globale du problème. Néanmois, elle n'est pas basée sur une logique géométrique de projection sur un sous-espace, conservant certaines propriétés des données : son premier objectif est de conserver des valeurs positives pour faciliter l'interprétation liée à des données non numériques. De plus, elle requiert que la matrice X ne contienne que des valeurs positives : cela restreint l'utilisation de cette méthode à certains types de problèmes ou requiert une normalisation adaptée.

Pour appliquer cette méthode, on choisit un entier r petit devant n et p. La factorisation permettra d'approximer les données X tel que :

$$X \simeq W.H \tag{5.19}$$

où $W(n \times r)$ et $H(r \times p)$ sont deux matrices ne contenant que des valeurs positives.

Cette factorisation est en fait un problème d'optimisation que l'on peut écrire comme la minimisation de la somme des écarts quadratiques entre la matrice des données X et son approximation $W.H$ avec une éventuelle pénalisation de régularisation. La solution optimale obtenue n'est qu'un optimum local car le critère n'est pas convexe, et dépend donc de l'initialisation de la méthode.

Les lignes de la matrice W peuvent être interprétées comme les projections des données dans un sous-espace $\mathbb{R}^p$. Les vecteurs de la nouvelle base sur laquelle sont projetées les données sont les colonnes de $H^T(HH^T)^{-1}$.

5.9 Projections aléatoires

Les méthodes de projections aléatoires sont très surprenantes par comparaison aux méthodes précédentes qui construisaient la projection afin de conserver certaines propriétés des données. Le but premier des méthodes de projections aléatoires est de réduire drastiquement la taille de l'espace de travail. Leur idée repose sur le lemme de Johnson-Lindenstrauss, selon lequel des données dans un espace de très grande dimension peuvent être projetées sur un espace de plus faible dimension tout en préservant les distributions de distances : ceci s'applique aux problèmes en très grandes dimensions.

Pour une matrice de données X (n données $\times$ p variables avec p grand), sa projection sur une base de vecteurs aléatoires s'écrit :

$$X_{proj} = X.R \tag{5.20}$$

où R est la matrice aléatoire ($p \times r$ avec $r < p$) dont les colonnes sont les vecteurs aléatoires unitaires de la nouvelle base de projection.

En pratique, la matrice R n'est pas totalement aléatoire. La première colonne de R est choisie aléatoirement et normalisée pour former un vecteur unitaire. La seconde colonne est choisie aléatoirement pour être orthogonale au premier vecteur, puis normalisée pour être unitaire. La troisième colonne est choisie aléatoirement dans le sous-espace orthogonal aux deux premiers vecteurs et normalisée pour être unitaire. Toutes les colonnes de R suivent cette même logique. Il en résulte que la base de vecteurs aléatoires est tout de même orthonormale.

La règle d'Achlioptas permet également de remplir la patrice R plus simplement :

$$R_{i,j} = \sqrt{3} \times \begin{cases} +1 & \text{avec la probabilité } \frac{1}{6} \\ 0 & \text{avec la probabilité } \frac{2}{3} \\ -1 & \text{avec la probabilité } \frac{1}{6} \end{cases} \tag{5.21}$$

5.10 Plongement local linéaire

Le plongement local linéaire (LLE, *Local Linear Embedding*) est une méthode de projection pour passer de l'espace original de dimension p vers un autre espace de dimension r. Ce plongement vise à préserver les distances dans un voisinage donné. Elle consiste en une série de projections locales linéaires qui sont ensuite assemblées pour reconstruire une projection globale non-linéaire.

Les étapes successives sont :

— On recherche tout d'abord les plus proches voisins $x^{(j)}$ d'un point donné $x^{(i)}$ dans l'espace original de dimension p.

— On identifie alors des poids w_{ij} par optimisation pour que le point $x^{(i)}$ puisse être approximé par une moyenne pondérée de ses plus proches voisins, en contraignant ces poids tels que $\sum_j w_{ij} = 1$:

$$w_{ij} = argmin_{w_{ij}} \sum_j \left(x^{(i)} - w_{ij}x^{(j)}\right)^2 \text{ soumis à } \sum_j w_{ij} = 1 \tag{5.22}$$

— Les poids w_{ij} étant désormais connus pour tous les points i, on cherche ensuite à reconstruire des points $y^{(i)}$ dans un nouvel espace de dimension r tout en imposant la matrice des poids w_{ij} :

$$y^{(i)} = argmin_{y^{(i)}} \sum_i \left(y^{(i)} - w_{ij}y^{(j)}\right)^2 \tag{5.23}$$

Puisque les points $y^{(i)}$ dans $\mathbb{R}^r$ et les points $x^{(i)}$ dans $\mathbb{R}^p$ possèdent la même matrice de poids w_{ij}, ils possèdent le même voisinage.

Géométriquement, on peut interpréter ce plongement local linéaire comme l'applatissement d'une surface. Si les données originales sont dans un espace à 3 dimensions mais organisées sur une surface courbe 2D dans cet espace, alors le plongement local linéaire peut être vu comme le déroulement de cette surface courbe sur une surface plane 2D. C'est de cette manière qu'on crée une carte géographique à partir de la surface courbe de la Terre.

5.11 Plongement spectral

Comme les méthodes spectrales en clustering (section 4.10), les méthodes spectrales de réduction de dimensionnalité font référence au spectre des valeurs propres d'une matrice particulière, en l'occurrence la matrice Laplacienne d'un graphe construit à partir des données.

Les étapes d'une méthode de plongement spectral sont :

— Les données $x^{(i)}$ dans $\mathbb{R}^p$ sont traitées, en fonction de leur voisinage, pour construire une matrice de poids W $(n \times n)$, calculée selon un critère de distance pour attribuer des poids faibles aux liaisons entre points très éloignés.
— On construit une matrice diagonale de poids, D, telle que $D_{ii} = \sum_j W_{ji}$
— La matrice Laplacienne est calculée : $L = D - W$. Cette matrice est symétrique définie positive et toutes ses valeurs propres sont positives. On cherche ensuite les valeurs propres et les vecteurs propres de :

$$Lv = \lambda Dv \tag{5.24}$$

— On sélectionne le nombre de dimensions r de l'espace latent de projection.
— On classe les valeurs propres par ordre croissant et on sélectionne les $r + 1$ premières valeurs. On laisse de côté la valeur propre λ_0 qui est égale à 0. Il reste :

$$\begin{cases} Lv_1 = \lambda_1 Dv_1 \\ Lv_2 = \lambda_2 Dv_2 \\ \dots \\ Lv_r = \lambda_r Dv_r \end{cases} \tag{5.25}$$

— Les vecteurs propres peuvent alors être utilisés pour connaître les nouvelles coordonnées d'une donnée i dans l'espace de dimension réduite :

$$x^{(i)} \rightarrow (v_1(i); v_2(i); \dots; v_r(i)) \tag{5.26}$$

5.12 Applications en Génie des Procédés

5.12.1 Recherche de descripteurs

Himmelblau et al. (1991) utilisent des signaux sonores pour détecter des erreurs de fonctionnement de procédés et réduisent de 38 à 15 le nombre de variables afin d'alimenter un réseau de neurones.

Dal-Pastro et al. (2017) cherchent à réduire la dimension de données issues de spectres NIR : chaque spectre contient 950 longueur d'ondes. Ils parviennent par analyse en composantes principales à réduire à 2 composantes tout en préservant 95 % de la variance globale.

5.12.2 Chimie et synthèse

Nandy et al. (2018) utilisent une méthode de forêts aléatoires pour réduire la dimension de leur problème de découverte de complexes de métaux de transition : de 150 caractéristiques moléculaires par complexe (pour 874 complexes différents), ils passent à moins de 30 descripteurs et divisent donc par 5 leur espace de travail.

5.12.3 Matériaux

Corma et al. (2005) cherchent à extraire des descripteurs spectraux en appliquant l'ACP à des spectres de diffraction de rayons X de matériaux catalytiques.

Ras et al. (2010) apliquent une analyse en composantes principales à des données issues de 576 tests catalytiques de conversion de biomasse : chaque donnée est initialement caractérisée par 8 grandeurs. Ils parviennent à réduire les données à seulement 2 composantes principales tout en expliquant 60

Fernandez et Barnard (2016) combinent une réduction de dimension par ACP avec un partitionnement par K-moyennes et plusieurs méthodes de classification pour identifier les matériaux MOFs les plus appropriés parmi une banque de 82 000 formulations afin d'optimiser les capacités d'adsorption de CO_2 et N_2.

Rothenberg (2008) passe en revue les différentes méthodes utilisées en catalyse, notamment l'ACP.

Fernandez et al. (2017) utilisent une base de données de nanoparticules catalytiques de platine pour identifier les meilleures géométries de particules : chacune est caractérisée par 18 propriétés géoémtriques. Avant de régresser les données ils appliquent une réduction de dimensionnalité et parviennent à identifier deux descripteurs de forme, qui serviront ensuite à alimenter des réseaux de neurones.

5.12.4 Simulation de réacteurs et de procédés

Kramer (1991) teste différentes structures de réseaux auto-encodeurs pour réduire l'espace des variables lors de la simulation d'un réacteur de synthèse en réacteur batch. Il fait varier le nombre de neurones dans les couches de codage et décodage allant jusqu'à 100-10-2-10-100.

5.12.5 Détection et diagnostic de fautes et dérives

Chiang et al. (2000) utilisent l'ACP, la DPLS et l'analyse discriminante de Fisher pour réduire la dimensionnalité d'un problème de détection de fautes sur un procédé Tennessee Eastman. Leur objectif est de réduire la dimension tout en préservant la distinguabilité des classes d'erreur dans l'espace réduit.

Onel et al. (2018) utilisent l'analyse en composantes principales et des méthodes SVM à noyaux non-linéaires pour identifier des descripteurs représentatifs de données issues de 22000 simulations de procédés batch catégorisées selon 15 types d'anomalies de fonctionnement.

Yoo (2020) développe une méthode améliorée pour la détection d'anomalie en combinant des méthodes de normalisation de variables, de réduction de dimension, de clustering et de recherche de corrélations.

Chapitre 6

Classification

Les méthodes de classification sont des méthodes d'apprentissage supervisé où l'étiquette des données symbolise la classe à laquelle appartient la donnée. On distingue parfois les méthodes de classification binaire, où les données appartiennent à 2 classes représentées par des étiquettes `0` ou `1` (ou préférentiellement `-1` et `1` pour parler de données négatives et positives), et les méthodes multiclasse où chacune des données peut appartenir à une classe parmi N représentées par des étiquettes `1` à `N`, par exemple. Les étiquettes de classe peuvent arbitrairement être représentées par des entiers ou tout autre type de variable. Chaque méthode peut généralement être adaptée à des problèmes binaires ou multiclasse.

x_1	x_2	x_3	y
15.09	68.59	610.5	0
18.68	66.19	1109.8	1
1.34	61.29	993.0	0
19.58	94.04	1374.4	1
⋮	⋮	⋮	
2.08	63.99	30.8	0
7.35	70.07	1324.1	1
10.51	67.20	1281.0	1

x_1	x_2	x_3	y
15.09	68.59	610.5	2
18.68	66.19	1109.8	1
1.34	61.29	993.0	3
19.58	94.04	1374.4	1
⋮	⋮	⋮	
2.08	63.99	30.8	3
7.35	70.07	1324.1	4
10.51	67.20	1281.0	1

TABLE 6.1 – Exemples de données étiquetées en vue de leur classification binaire (gauche) et multiclasse (droite).

L'idée générale des méthodes de classification consiste à être capable d'assigner à chaque nouvelle donnée la classe dans laquelle elle doit être rangée sur la base de la seule connaissance de ses variables. En mode de test, ces méthodes reçoivent donc les variables de la donnée et lui affectent une étiquette. Pour parvenir à cette discrimination, la méthode a dû être préalablement entrainée sur des données déjà étiquetées : l'entrainement permet à la méthode d'identifier des frontières entre les classes, ces frontières pouvant être nettes ou floues au sens d'une probabilité de certitude de classement.

En Génie des Procédés, des exemples de classification concernent (voir section 6.8) :

— les types d'anomalies que subit un procédé en fonction des variables opératoires qui le décrivent,
— les types de matériaux ou de formulations de produits qui permettent d'obtenir des classes de performances ou de propriétés d'usage,
— les types de régimes d'écoulement dans des systèmes multiphasiques en fonction de nombres adimensionnels.

6.1 Critères de qualité d'une classification

Il existe plusieurs critères qualitatifs et quantitatifs pour estimer la qualité d'une méthode de classification. On résumé ici les principales caractéristiques de ces critères.

6.1.1 Matrice de confusion

Lorsqu'une méthode de classification décide d'affecter une donnée à une classe, elle peut se tromper. Dans le cas d'une classification binaire, il existe une certaine probabilité qu'elle affecte une donnée dans la classe `0` alors qu'elle aurait dû être affectée à la classe `1` (ou inversement). De façon tout à fait analogue aux tests d'hypothèses en statistiques, on définit ces types d'erreurs dans une matrice de confusion (Tableau 6.2), dans laquelle on distingue :

— Les "Vrais Négatifs (VN)" sont les classifications justes dans la classe `0`. La notion négatif/positif est issue du diagnostic médical où un patient est porteur (classe `1`) ou non (classe `0`) d'une pathologie.
— Les "Faux Positifs (FP)" sont des erreurs où la donnée est classée `1` au lieu de `0`. En statistiques (tests d'hypothèses), ces faux positifs désignent l'erreur de première espèce, α, aussi appelée "niveau de signification".
— Les "Faux Négatifs (FN)" sont des erreurs de seconde espèce, β.
— Les "Vrais Positifs (VP)" sont des données correctement classées dans la classe `1`.

		Classe réelle	
		0	1
Classe prédite	0	Vrai Négatif (VN)	Faux Négatif (FN)
	1	Faux Positif (FP)	Vrai Positif (VP)

TABLE 6.2 – Matrice de confusion d'une classification binaire.

6.1.2 Spécificité, sensibilité, erreurs de première et seconde espèce

La sensibilité d'un test, médical par exemple, est la capacité de ce test à détecter effectivement une pathologie lorsqu'elle est présente : il s'agit donc du taux de vrais positifs par rapport à tous les échantillons effectivement positifs. Elle est le complémentaire à 1 de l'erreur de seconde espèce β :

$$\text{Sensibilité} = \frac{VP}{VP + FN} = 1 - \beta \tag{6.1}$$

La spécificité du test est sa capacité à ne pas fournir un résultat positif quand il n'y a pas de raison d'être positif : le test est spécifique dans le sens où il n'est pas perturbé par d'autres éléments. La spécificité est le taux de vrais négatifs parmi tous les échantillons effectivement négatifs. Elle est le complémentaire de l'erreur de première espéce α.

$$\text{Spécificité} = \frac{VN}{VN + FP} = 1 - \alpha \tag{6.2}$$

Ces grandeurs, ainsi que la matrice de confusion, ne peuvent être utilisées que sur des données supervisées car il est nécessaire de connaître les vraies de chaque données pour savoir si la méthode fournit une bonne ou une mauvaise réponse.

6.1.3 Courbe ROC et surface AUC

La courbe ROC (*Receiver-Operator Characteristics*) et la surface AUC (*Area Under the Curve*) sont deux critères très utilisés pour qualifier une méthode de classification ou comparer plusieurs méthodes.

Pour comprendre leur construction et leur pertinence, il est plus pratique de se baser sur un exemple. On considèrera ici 20 données dépendant d'une variable x et étiquetée 0 ou 1 selon leur classe (Figure 6.1 et Tableau 6.3). On peut tout de suite remarquer que les classes ne sont pas parfaitement disjointes puisque 2 données étiquetées 0 se trouvent parmi des données étiquetées 1.

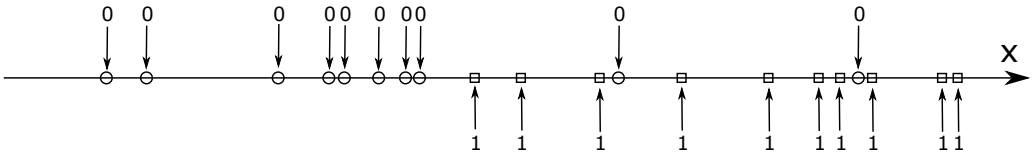


FIGURE 6.1 – Données étiquetées pour une classification binaire.

Comme les étiquettes 0 sont essentiellement associées à des valeurs de x faibles, une méthode de classification possible de ces données consiste à choisir une valeur seuil de x, x_{seuil}, et à décider que toutes les données vérifiant $x^{(i)} < x_{seuil}$ seront étiquetées 0 alors que celles vérifiant $x^{(i)} > x_{seuil}$ seront étiquetées 1.

Pour chaque valeur de x_{seuil}, on peut ainsi prédire les étiquettes pour chaque donnée et calculer les éléments de la matrice de confusion : VN, VP, FN et FP, ainsi que le taux de faux positifs, $TFP = \frac{FP}{FP+VN}$, et le taux de vrais positifs, $TVP = \frac{VP}{VP+FN}$. Par exemple, si on choisit $x_{seuil} = 1$ pour les données du Tableau 6.3, on va prédire l'étiquette 0 pour 13 données et l'étiquette 1 pour les 7 autres. Si on compare ces prédictions avec les étiquettes réelles, on trouve : $VN = 9$, $VP = 6$, $FN = 4$ et $FP = 1$, soit $\frac{FP}{FP+VN} = 0.1$ et $\frac{VP}{VP+FN} = 0.6$. Cette

x	y
-1.5850	0
-1.4226	0
-0.8900	0
-0.6851	0
-0.6225	0

x	y
-0.4834	0
-0.3751	0
-0.3185	0
-0.0951	1
0.0917	1

x	y
0.4100	1
0.4862	0
0.7418	1
1.0930	1
1.2959	1

x	y
1.3827	1
1.4564	0
1.5118	1
1.7949	1
1.8581	1

TABLE 6.3 – Exemples de 20 données, avec leurs étiquettes vraies, en vue de leur classification binaire.

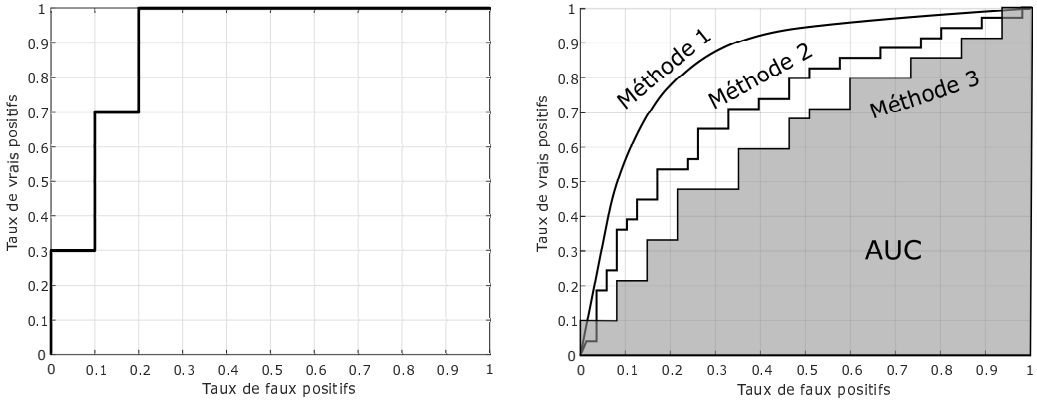


FIGURE 6.2 – Courbe ROC de l'exemple (gauche) et formes de courbes ROC possibles et visualisation de la surface AUC (droite).

application permet de placer un point sur une courbe $TVP = f(TFP)$, la fameuse courbe ROC. En faisant varier la valeur de x_{seuil}, on peut tracer toute la courbe $TVP = f(TFP)$ (Figure 6.2, gauche).

Pour toutes les valeurs de x_{seuil} inférieures à -1.5850, cette méthode de classification prédit que toutes les étiquettes sont égales à 1, donc $VN = 0$, $VP = 10$, $FN = 0$ et $FP = 10$, et $TFP = 1$ et $TVP = 1$, donc la courbe passe par le point $(1; 1)$ de ce graphe. Lorsqu'on augmente le seuil, à chaque fois qu'on dépasse le $x^{(i)}$ d'une variable, son étiquette prédite passe de 1 à 0, donc le nombre de prédictions positives baisse d'une unité : selon que l'étiquette vraie de cette donnée est 0 ou 1, c'est soit le nombre de vrais positifs VP qui baisse soit le nombre de faux positifs FP qui baisse. Dans tous les cas, TFP ou TVP baisse : la courbe ROC est donc monotone. A l'extrême, si x_{seuil} est supérieure à 1.8581, toutes les données sont étiquetées 0 donc $FP = 0$ et $VP = 0$, donc $TFP = 0$ et $TVP = 0$, donc la courbe passe par $(0; 0)$.

La courbe ROC passe donc toujours par les points $(0; 0)$ et $(1; 1)$ et est monotone en escalier entre ces points. La courbe est d'autant plus lisse que le nombre de données est grand.

Une méthode de classification parfaite permettrait de supprimer toutes les erreurs de classification, on aurait donc $FP = 0$ et $FN = 0$. La courbe passerait donc par le point $(0; 1)$ et maximiserait l'aire sous la courbe. C'est pour cette raison que l'aire sous la courbe (AUC, *Area Under the Curve*, Figure 6.2 (droite)) est un critère important de comparaison des méthodes : plus cette aire est grande, meilleure est la méthode de classification.

6.2 Méthode des K plus proches voisins

La méthode des K plus proches voisins est une méthode simpliste de classification. Elle ne nécessite pas de phase d'entrainement et permet de classifier chaque nouvelle donnée en fonction d'un jeu de données déjà étiquetées. Comme son nom l'indique, le choix d'affectation à une classe se base exclusivement sur les données les plus proches de la nouvelle donnée à classer (Figure 6.3).

Pour une nouvelle donnée c à classer, on cherche les K données $x^{(i)}$ les plus proches selon une distance $d(c, x^{(i)})$ dans l'espace des variables. On notera $1 \leq k \leq K$ les indices de ces voisins. Pour affecter une étiquette à la nouvelle donnée c, on cherche l'étiquette la plus fréquente parmi les étiquettes des plus proches voisins $y^{(1 \leq k \leq K)}$. Dans le cas de la Figure 6.3, si on s'intéresse aux 3 plus proches voisins, on trouve deux ronds et un carré. On affectera donc l'étiquette $\circ$ à la nouvelle donnée. Néanmoins, celle-ci ne sera pas incluse dans le jeu de données de référence pour les affectations suivantes car elle induirait un biais de méthode.

Parmi les caractéristiques de cette méthode, on peut citer :

— La méthode ne cherche pas à définir de frontières entre les domaines de prédominance des classes, mais on peut tout de même les identifier. Puisque la comparaison d'un donnée se fait par un calcul de distance minimale avec ses voisins, les frontières sont obtenues par un diagramme de Voronoï. Si $K = 1$, chaque

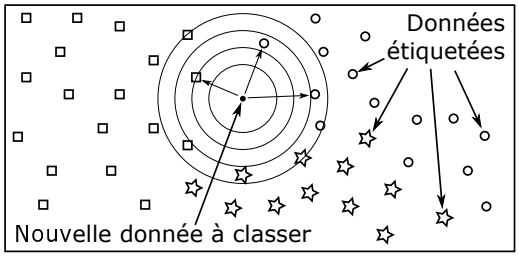


FIGURE 6.3 – Illustration du principe de la méthode des K plus proches voisins.

donnée du jeu de référence définit une zone d'attraction qui est un polygone de Voronoï. Si K augmente, les polygones fusionnent et on obtient des frontières linéaires par morceaux mais dont la forme peut devenir plus complexe. Plus K est grand, plus les frontières sont lisses.

— Puisque cette méthode est liée à une notion de similarité par distance, elle est directement impactée par le fléau de la dimensionnalité : plus la dimension p de l'espace des variables est grande, moins cette notion de distance est pertinente car tous les points sont éloignés les uns des autres. La "proximité" n'assure donc plus la ressemblance.

— Le nombre de plus proches voisins K est le seul hyper-paramètre de la méthode. Il est donc nécessaire de le fixer par optimisation ou de le choisir selon des critères rationnels. Si K est petit, la méthode est très sensible au bruit. Si K est du même ordre de grandeur que le nombre total de données n, alors la méthode prédira toujours la classe dominante, quelle que soit la position de c par rapport aux données. Une heuristique préconise de choisir $K \simeq \sqrt{n}$.

— Le calcul des distances entre la nouvelle donnée c et l'ensemble des points de l'espace pour identifier les plus proches voisins est chronophage.

On peut également citer deux variantes de cette méthode :

— Variante des ϵ-voisins : le nombre de voisins n'est pas fixé arbitrairement, on choisit toutes les données dans un ϵ-voisinage autour de la nouvelle donnée c. Cela permet de fiabiliser la décision au coeur des éventuels clusters. Néamoins, dans les zones vides, il est probable que ce voisinage soit vide et une méthode alternative doit alors être mise en oeuvre.

— Variante des K voisins pondérés : dans la méthode de base, l'affectation d'étiquette se fait par vote de la majorité, c'est l'étiquette la plus fréquente qui est retenue. La variante des voisins pondérés ajoute un poids dans ce vote, inversement proportionnel à la distance : les voisins les plus proches ont plus de poids que les voisins éloignés.

6.3 SVM - Séparateur à Vaste Marge

L'abbréviation SVM signifie originellement "Support Vector Machine" et se traduit naturellement par "Machine à vecteur de support", ou plus spécifiquement "Séparateur à Vaste Marge". Il s'agit d'une famille de méthodes de classification très puissantes et adaptées pour un grand nombre de cas de figures. Les variantes décrites ci-dessous permettent de classifier des données linéairement séparables, disjointes ou avec recouvrement, ou non linéairement séparables. On présentera ici ces méthodes pour des variables binaires, pouvant donc être classées selon deux catégories.

6.3.1 Cas linéairement séparable : SVM à marge rigide

La Figure 6.4 présente un ensemble de données $D = \{x_j^{(i)}, 1 \leq j \leq 2; y^{(i)}\}(1 \leq i \leq n)$, appartenant à deux classes ($\times$ pour les étiquettes `-1` et $+$ pour les `+1`), "linéairement séparables", c'est-à-dire qu'il est possible de séparer/classer ces données en définissant une frontière linéaire : dans $\mathbb{R}^2$, cette frontière est une droite, comme sur la figure, et dans $\mathbb{R}^p$, cette frontière serait un hyperplan de dimension $p-1$.

La Figure 6.4 (gauche) montre qu'il est souvent possible de définir plusieurs frontières (en fait, une infinité) qui divisent effectivement l'ensemble des données mais qui passent plus ou moins près de certaines données. Pour chacune de ces frontières, qu'on appellera "séparateur", on peut définir une "marge", qui est la distance entre le séparateur et la donnée la plus proche. Cette donnée la plus proche peut appartenir à l'une ou l'autre des classes. Un bon séparateur ne devrait pas favoriser l'une ou l'autre des classes et devrait donc être équidistant des deux familles de données. L'idée de base des SVM consiste à optimiser le séparateur pour qu'il maximise la marge sans favoriser l'une ou l'autre des classes.

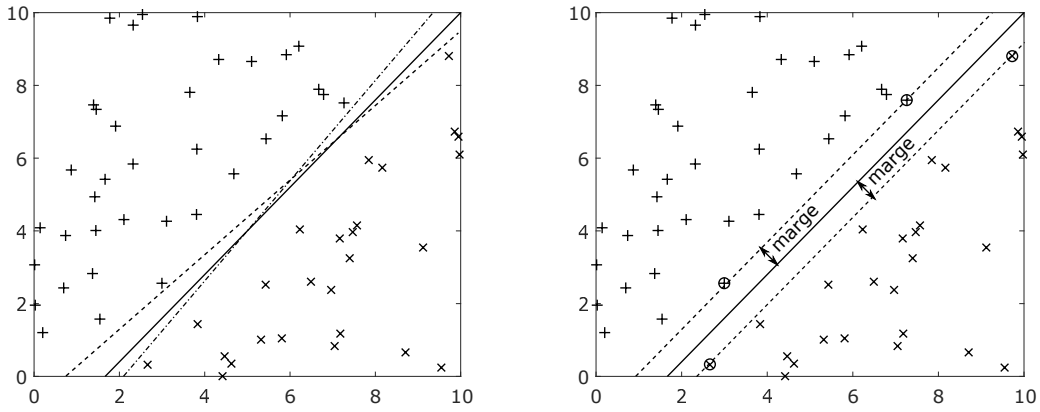


FIGURE 6.4 – Représentation d'un ensemble de données (linéairement séparables) avec plusieurs séparateurs linéaires (gauche) et le séparateur de marge maximale (droite).

La Figure 6.4 (droite) montre le séparateur optimal qui maximise la marge. Le séparateur H apparaît en trait continu. Il est entouré des deux séparateurs H^- et H^+ en pointillés, qui sont équidistants de H et se confondent avec certaines données de la classe négative et de la classe positive ($\otimes$ et $\oplus$). Ces données particulières sont appelées "vecteurs de support" car ce sont les données, donc des vecteurs dans $\mathbb{R}^2$ ou $\mathbb{R}^p$, qui supportent les hyperplans H^- et H^+ qui définissent le séparateur et sa marge. La zone entre H^- et H^+ est la zone d'indécision.

Mathématiquement, un hyperplan dans $\mathbb{R}^p$ est décrit par une équations de la forme :

$$w_1x_1 + w_2x_2 + ... + w_nx_n + b = 0 \tag{6.3}$$

Si on note w le vecteur des poids (aussi appelé vecteur directeur) et x le vecteur $[x_1\ x_2\ \dots\ x_n]^T$, cette équation devient :

$$w^Tx + b = 0 \tag{6.4}$$

Les degrés de liberté sur w et b permettent alors de définir le séparateur par ses trois hyperplans :

$$\begin{cases} H : w^Tx + b = 0 \\ H^- : w^Tx + b = -1 \\ H^+ : w^Tx + b = 1 \end{cases} \tag{6.5}$$

Toutes les données négatives satisfont $w^Tx^{(i)}+b \leq -1$ et toutes les données positives satisfont $w^Tx^{(i)}+b \geq 1$. Toutes les données satisfont donc le fait que le produit de leur étiquette $y^{(i)}$ par leur évaluation par les équations du séparateur $w^Tx^{(i)} + b$ est donc positif et même supérieur à 1 : $y^{(i)}.(w^Tx^{(i)} + b) \geq 1$. On peut également calculer la marge, notée γ, du séparateur à partir de w : $\gamma = \frac{1}{\|w\|_2}$.

En pratique, les caractéstiques w et b du séparateur linéaire optimal sont calculées par maximisation de la marge (donc la minimisation de son inverse), soumise à la bonne évaluation du signe des étiquettes :

$$w, b = \arg \min_{w\in\mathbb{R}^p, b\in\mathbb{R}} \frac{1}{2} \|w\|_2^2 \qquad \text{soumis à} \qquad y^{(i)}.(w^Tx^{(i)} + b) \geq 1 \quad (1 \leq i \leq n) \tag{6.6}$$

La présence du facteur $\frac{1}{2}$ et du carré sur $\|w\|_2^2$ ne sont pas nécessaires mais simplifient la réécriture du problème sous sa forme duale, qui est ici plus simple à résoudre (Azencott, 2019).

Ce problème d'optimisation ne possède de solutions que si les données sont linéaires séparables et donc totalement disjointes.

6.3.2 Cas non linéairement séparable : SVM à marge souple

La Figure 6.5 présente deux classes de données non linéairement séparables, et un séparateur linéaire. Les symboles $\times$ et $+$ représentent les données correctement classées. Les symboles $\otimes$ et $\oplus$ représentent les vecteurs de support sur lesquels s'appuient H^- et H^+. Dans la zone d'indécision entre H^- et H^+, les données entourées d'un carré sont mal classifiées. Les données entourées d'un rond sont bien classées mais dans la zone d'indécision : il aurait été possible de choisir un séparateur pour lequel elles n'auraient pas été dans la zone d'indécision, mais sa marge aurait été beaucoup plus petite.

Le séparateur à marge souple commet donc inévitablement des erreurs car certaines données ne peuvent pas être classées linéairement. L'optimisation de son réglage est donc un compromis entre la maximisation de

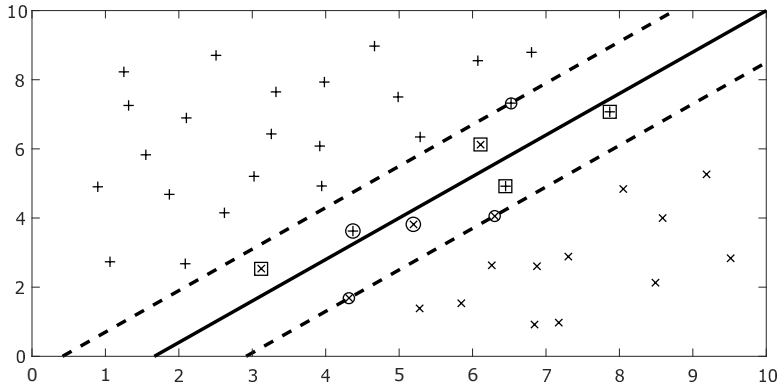


FIGURE 6.5 – Représentation d'un ensemble de données (non linéairement séparables).

la marge et la minimisation du coût qu'on accepte de payer pour les erreurs qu'il commet. Le critère global à minimiser prend donc la forme générale :

$$w, b = \arg \min_{w \in \mathbb{R}^p, b \in \mathbb{R}} \left(\frac{1}{2} \|w\|_2^2 + C \sum_{i=1}^{n} L\Big(w^T x^{(i)} + b, y^{(i)} \Big) \right) \tag{6.7}$$

où C est un hyperparamètre pour pondérer le poids relatif de la marge et du coût des mauvaise classifications, et $L()$ est une fonction de coût qui dépend de l'étiquette vraie de la donnée $y^{(i)}$ et de l'estimation de cette étiquette $w^T x^{(i)} + b$. L'hyperparamètre C donne de la souplesse dans la pondération des deux effets, d'où le nom de "marge souple". La fonction de coût L doit défavoriser les mauvaises classifications sans que les classifications correctes n'impactent le critère, c'est donc généralement une fonction asymétrique : pour une classification binaire, la prédiction est fausse quand $y^{(i)}$ et $w^T x^{(i)} + b$ sont de signes opposés, donc quand leur produit est négatif. Une constante est ajoutée pour tenir compte de la zone d'indécision. On utilise souvent l'erreur "hinge" :

$$L\Big(w^T x^{(i)} + b, y^{(i)} \Big) = L_{hinge}\Big(w^T x^{(i)} + b, y^{(i)} \Big) = \max\Big(0; 1 - y^{(i)} (w^T x^{(i)} + b) \Big) \tag{6.8}$$

6.3.3 Cas non-linéaire : SVM à noyau

Dans beaucoup de cas, la frontière entre les classes ne peut pas être approximée par un séparateur linéaire. La Figure 6.6 présente des données non linéairement séparables ($\circ$ et $+$), soit dans leur espace original $(x_1; x_2)$ (gauche), soit dans un espace de redescription plus adapté $(x_1^2; x_2^2)$ (droite). Alors que les données ne sont pas linéairement séparables dans l'espace original, on voit immédiatement qu'elles sont linéairement séparables dans l'espace de redescription. La classification peut donc être réalisée dans l'espace de redescription.

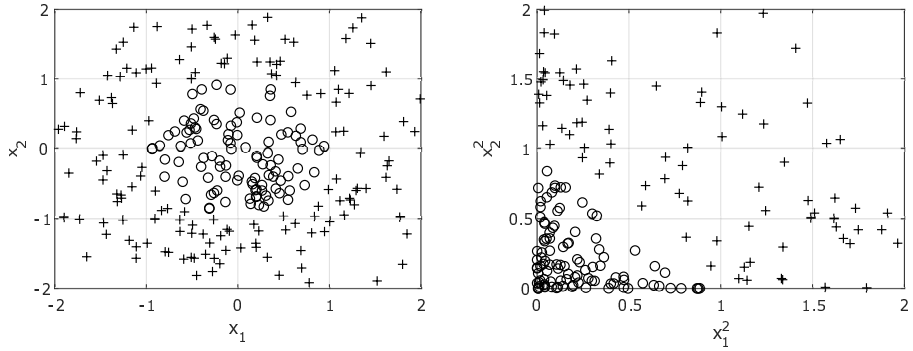


FIGURE 6.6 – Représentation d'un ensemble de données (non linéairement séparables), présentées dans l'espace original des variables x_1 et x_2 (gauche) et dans un espace de redescription x_1^2 et x_2^2.

Pour cet exemple, le changement d'espace, depuis l'espace original vers l'espace de redescription, se fait par une application :

$$\begin{aligned} \phi(x) : \quad \mathbb{R}^2 \quad &\to \mathbb{R}^2 \\ (x_1; x_2) \quad &\to (x_1^2; x_2^2) \end{aligned} \tag{6.9}$$

Tout le calcul du séparateur linéaire dans l'espace de redescription et de la fonction de décision dans l'espace original peut ensuite se faire sur la base d'une fonction qu'on appelle le "noyau" et qui est en fait un produit scalaire entre les images par l'application ϕ :

$$\begin{aligned} k(x,x') : \quad \mathbb{R}^p \times \mathbb{R}^p \quad &\to \mathbb{R} \\ x, x' \quad &\to \phi(x)^T.\phi(x') \end{aligned} \tag{6.10}$$

Mais il n'est finalement pas nécessaire de connaître l'application $\phi(x)$, il suffit de connaître le noyau $k(x,x')$, qui peut généralement s'exprimer en fonction de x et x' sans faire appel à ϕ.

Toute cette transformation, appelée "astuce du noyau", permet donc d'appliquer des méthodes de classification linéaires sur des données qui ne sont pas linéairement séparables, sans avoir à chercher le changement de variable non-linéaire nécessaire. Le lecteur intéressé trouvera plus de détails sur cette astuce dans Azencott (2019). Cette astuce peut être étendue à d'autres méthodes d'apprentissage comme les K-moyennes et l'ACP.

6.4 Arbres de décision

Les arbres de décision sont une méthode de classification hiérarchique. Ils sont constitués d'une série de tests conditionnels, qui définissent les embranchements de l'arbre depuis la racine (en haut) jusqu'aux feuilles (en bas) qui désignent les classes "décidées" (Figure 6.7, gauche). Chaque test conditionnel effectue une partition de l'espace des variables (Figure 6.7, droite) et permet de borner plus précisément une région de prédominance d'une étiquette particulière.

Ils permettent de traiter des problèmes binaires aussi bien que multiclasse, sur des variables/critères continus ou discrets, et peuvent également représenter des classes multimodales, comme les $\times$, les $\circ$ et les $\square$ qui se trouvent dans des zones disjointes du plan $(x_1; x_2)$. Ils peuvent également être utilisés pour construire des régressions.

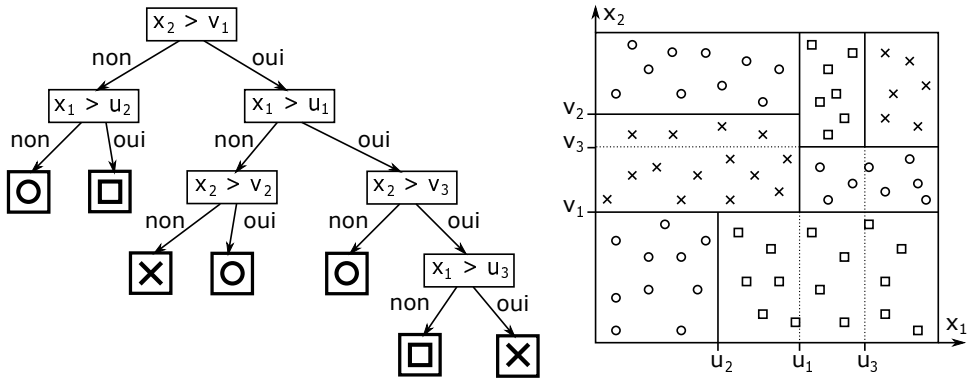


FIGURE 6.7 – Représentation d'un arbre de décision (gauche) et de la classification de données étiquetées qu'il permet dans $\mathbb{R}^2$.

Dans l'exemple de la Figure 6.7, l'arbre parvient à identifier des régions de $\mathbb{R}^2$ où une seule étiquette est présente, mais chaque région pourrait contenir un mélange d'étiquettes. La classe "décidée" pour cette région serait alors choisie comme étant l'étiquette majoritaire. Si l'on note $\delta(y^{(i)}, c)$ le Kronecker de l'étiquette d'une variable $\{x^{(i)}; y^{(i)}\}$ et d'une classe c parmi C classes possibles ($\delta(y^{(i)}, c) = 1$ si $y^{(i)} = c$; $\delta(y^{(i)}, c) = 0$ si $y^{(i)} \neq c$), alors l'étiquette majoritaire d'une région R est :

$$c_{majoritaire}(R) = \arg \max_{c=1,\dots,C} \sum_{x^{(i)} \in R} \delta(y^{(i)}, c) \tag{6.11}$$

6.4.1 Critères d'uniformité des régions

Chaque test conditionnel d'un arbre vise à partitionner une région afin que les données contenues dans chacune des deux sous-régions soient plus homogènes que les données de la région initiale. Il existe donc plusieurs critères pour quantifier l'uniformité, ou la "pureté", des données contenues dans une région.

Si une région R contient n_R données $\{x^{(i)}; y^{(i)}\}$, alors on peut calculer la proportion de données représentant chaque classe c parmi les C classes possibles tel que :

$$p_c(R) = \frac{1}{n_R} \sum_{x^{(i)} \in R} \delta(y^{(i)}, c) \qquad \text{avec} \qquad 1 \leq c \leq C \tag{6.12}$$

La classe majoritaire est donc celle dont la proportion est la plus élevée :

$$c_{majoritaire}(R) = \arg \max_{c=1,\dots,C} p_c(R) \tag{6.13}$$

Impureté par erreur de classification

La proportion des données qui n'appartiennent pas à la classe majoritaire fournit l'erreur de classification dans cette région et permet de définir l'impureté de cette région, $Imp(R)$:

$$Imp(R) = 1 - \max_{c=1,\dots,C} p_c(R) \tag{6.14}$$

Cette impureté est bornée entre 0 et 1. Elle est nulle si toutes les données de la région sont de la même classe. Par contre, si toutes les C classes sont équitablement représentées ($p_c(R) = \frac{1}{C}$), cette impureté vaut $1 - \frac{1}{C}$.

Impureté de Gini

L'impureté de Gini est une estimation de la probabilité qu'une donnée tirée aléatoirement parmi celles de la région soit mal classée si on lui attribue une étiquette choisie aléatoirement parmi les étiquettes des données présentes selon leur probabilité de présence. Si C classes sont présentes dans R, la probabilité qu'une donnée soit de la classe c est $p_c(R)$ et la probabilité qu'elle se retrouve mal classée est $1 - p_c(R)$. L'impureté de Gini est alors :

$$Imp(R) = \sum_{c=1}^{C} p_c(R)\big(1 - p_c(R)\big) \tag{6.15}$$

Si une seule classe est représentée dans la région R, alors cette impureté de Gini vaut 0. Si toutes les classes sont équitablement représentées, alors elle vaut $1 - \frac{1}{C}$.

Impureté par entropie croisée

La théorie de l'information de Shannon a formalisé le concept d'entropie pour des données. Cette entropie mesure le désordre parmi un ensemble de données. Elle s'écrit :

$$Imp(R) = -\sum_{c=1}^{C} p_c(R) \log_2\big(p_c(R)\big) \tag{6.16}$$

Si une seule classe est présente dans R, alors $p_c(R) = 1$ et l'entropie est donc nulle. Si toutes les classes sont équitablement représentées, alors $p_c(R) = \frac{1}{C}$ et l'entropie vaut $\log_2(C)$, donc 1 pour une classification binaire.

Toutes ces définitions ont en commun d'être toujours positives, et nulles lorsque toute les données de la région possèdent la même étiquette. La construction de l'arbre de décision vise donc à minimiser l'impureté dans chacune des régions finales.

6.4.2 Construction d'un arbre

Il existe plusieurs algorithmes de construction d'arbres de décision : ID3 (*Iterative Dichotomiser 3*), C4.5, C5 (successeurs d'ID3), CART (*Classification And Regression Tree*), etc.

L'idée générale de ces algorithmes consiste à construire itérativement les divers embranchements de l'arbre en choisissant à chaque fois la meilleure variable séparatrice pour construire le test conditionnel qui permettra de minimiser l'impureté des deux sous-régions définies par ce test. Des variantes binaires et discrètes de cette étape existent (Azencott, 2019), mais on présente ici le cas de variables réelles continues.

Pour un arbre donné (préexistant, ou initialisé par la simple racine), on souhaite remplacer une feuille par un nouveau test conditionnel. La feuille définissait une région R que l'on va diviser en deux sous-régions complémentaires R_1 et R_2 telles que $R = \bigcup(R_1; R_2)$. On note $\{x_j^{(i)}; y^{(i)}\}(1 \leq j \leq p)$ les n_R données incluses dans la région R de $\mathbb{R}^p$, et appartenant à C classes. La construction du test conditionnel consiste à choisir l'une des variables x_j d'indice j et une valeur discriminante s pour définir le test $x_j > s$. Les inconnues j et s doivent être optimisées pour minimiser l'impureté totale des deux sous-régions :

$$j^*, s^* = \arg\min_{j,s} \left(\frac{n_{R_1}}{n_R} Imp\big(R_1(j,s)\big) + \frac{n_{R_2}}{n_R} Imp\big(R_2(j,s)\big) \right) \tag{6.17}$$

En pratique, les algorithmes se distinguent par la méthode de détermination de j^* et s^*. Par exemple, l'algorithme $CART$ utilise une logique de force brute en testant tous les indices j possibles et toutes les valeurs de s possibles. Puisque le nombre de données n_R est fini, le nombre de valeurs de s à tester l'est aussi, on utilise en général les valeurs médianes des variables $x_j^{(i)}$ triées ($x_j^{(1)} < x_j^{(2)} < \dots < x_j^{(n_R)}$) : $s(j) \in \left\{ \frac{x_j^{(i)} + x_j^{(i+1)}}{2} \right\}$.

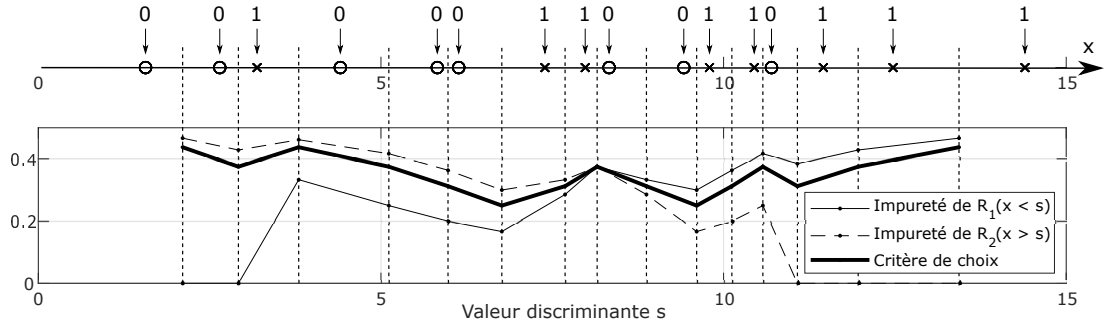


FIGURE 6.8 – Illustration du choix optimal d'un test conditionnel pour une classification binaire à 1 dimension.

La Figure 6.8 présente une illustration de cette construction d'un test conditionnel. 16 données sont réparties selon la variable x entre 0 et 15 et possèdent des étiquettes 0 ou 1 (8 données de chaque classe). On veut identifier le meilleur test conditionnel $x > s$ pour diviser en 2 groupes ces données en minimisant l'impureté des deux sous-régions calculée selon le taux d'erreur de classification. Les 15 valeurs discriminantes de s, i.e. les médianes entre points adjacents, apparaissent sous forme de pointillés verticaux. Chacune de ces valeurs permet de définir les 2 sous-régions ($R_1 : x < s$ et $R_2 : x > s$) et de calculer l'impureté de chaque sous-région et l'impureté globale. Les impuretés des régions sont effectivement nulles lorsqu'elles ne contiennent qu'une étiquette et tendent vers 0.5 lorsqu'elles contiennent presque toute les données. L'impureté globale, qui va permettre de choisir le test conditionnel optimal, présente, dans ce cas, deux minima globaux identiques (en $s = 6.76$ et $s = 9.60$) qui fournissent la même impureté 0.25. Ces deux partitions possibles sont donc équivalentes en termes de séparation des données binaires.

Cette construction itérative de l'arbre peut être poussée jusqu'à ce que chaque feuille ne contienne plus qu'une donnée. L'arbre contient alors $n_R - 1$ tests conditionnels. Ce cas de figure n'est généralement pas conseillé car il est synonyme de sur-apprentissage. On préfère en général arrêter l'ajout d'embranchements lorsque chacune des feuilles possède, soit une impureté inférieure à un seuil acceptable, soit un nombre minimum de données.

6.5 Réseaux de neurones

Les réseaux de neurones peuvent aussi être utilisés pour la classification de données. Leur logique de construction et d'entrainement est présentée en détails dans le Chapitre 7. Seules les spécificités de leur utilisation pour la classification sont présentées ici.

Puisqu'un réseau de neurones, ou même un simple perceptron monocouche, possède la capacité de corréler tout type de fonction non-linéaire en fonction de ses variables, $y = f(x_1, x_1, ..., x_p)$, cette capacité peut être exploitée pour prédire un étiquette de classification $y^{(i)}$ en fonction des variables $x^{(i)}$ d'une donnée. Pour la classification, certains hyper-paramètres du réseau doivent être choisis de façon à un ajustement adapté du réseau.

Classification binaire

Pour une classification binaire, les étiquettes $y^{(i)}$ des données sont 0 ou 1. Il semble donc assez logique d'utiliser, pour l'activation des neurones de la couche cachée, une fonction d'activation à seuil, de type Heaviside.

La Figure 6.9 (a) présente un exemple de deux séries de données de classe 0 ou 1 avec recouvrement partiel. Un unique neurone suffit à définir le saut pour représenter l'essentiel des étiquettes 0 et 1. Trois réglages différents des poids synaptiques et du biais du neurone caché sont présentés (courbes (1), (2) et (3)). Le réglage (1) échoue à prédire les étiquettes des données C et D. Le réglage (2) échoue à prédire les données A, B, C et D. Le réglage (3) échoue à prédire A, B et D. Numériquement, le réglage (1) semble le meilleur, pourtant la visualisation des données semble dire que le réglage (2) est une frontière plus juste pour délimiter l'ensemble des étiquettes 0 et 1. Sans même parler des problèmes d'entrainement que pose la fonction de Heaviside par annulation du gradient, il semble que cette fonction rende la définition d'une frontière très sensible à la position précise des données.

On lui préfère généralement la fonction sigmoïde, dont les valeurs asymptotiques sont aussi 0 et 1 mais qui assure une transition douce entre ces valeurs. Elle permet surtout d'extraire de son expression une probabilité d'appartenance à une classe. L'expression de la sortie y en fonction des entrées x de la couche d'entrée, des poids synaptiques w et du biais b du neurone caché, est de la forme :

$$y(x) = \frac{1}{1 + \exp(w.x + b)} \tag{6.18}$$

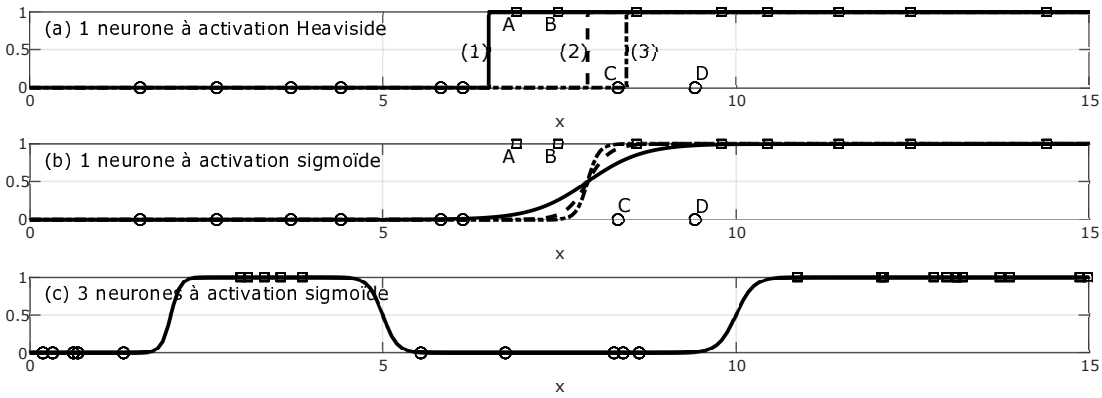


FIGURE 6.9 – Exemples de classifications binaires par réseaux de neurones simples, pour des données en 1 dimension.

La Figure 6.9 (b) présente 3 réglages possibles de poids w et du biais b de cette fonction sigmoïde, qui permettent d'adoucir la transition des étiquettes 0 vers les étiquettes 1.

Lorsque les classes sont multimodales en 1 dimension, ou en dimensions supérieures, le nombre de neurones doit être augmenté comme le montre la Figure 6.9 (c) sur laquelle 3 neurones sont nécessaires pour reproduire les 3 sauts entre les étiquettes 0 et 1. Il est important de préciser que l'activation sigmoïde est nécessaire également dans la couche de sortie pour assurer que la sortie globale soit bien entre 0 et 1.

La Figure 6.10 présente un exemple de classification binaire par réseau de neurones à 2 dimensions. La vue de gauche présente le plan $(x_1; x_2)$ vu du dessus avec les symboles $\circ$ et $\times$ pour les deux classes de données. La vue de droite présente la surface de prédiction des étiquettes interpolée dans le plan à partir de l'expression du réseau de neurones complet.

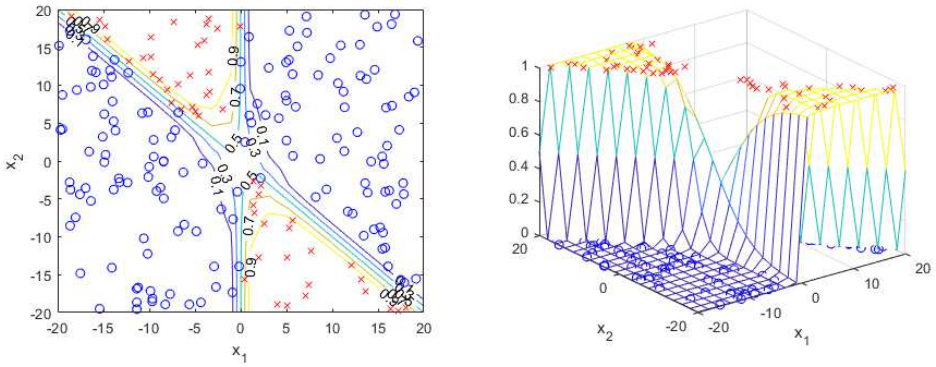


FIGURE 6.10 – Exemple de classification binaire par réseau de neurones en 2 dimensions.

Classification multi-classe

Dans le cas d'une clasification multi-classe, un unique neurone de sortie ne suffit par à représenter le choix d'une étiquette parmi C classes. La structure du réseau doit donc être modifiée pour contenir plusieurs neurones de sortie (Figure 6.11).

Afin que les neurones de sortie soient capables de désigner une classe sans confusion, ils doivent tous répondre 0, sauf un qui doit répondre 1 : celui-ci désigne alors la classe prédite. Les réponses des neurones de cette couche de sortie ne sont donc pas indépendantes. Pour cela, on utilise une fonction d'activation particulière, appelée fonction "Softmax", définie telle que :

$$\sigma(o_1, o_2, ..., o_C)_c = \frac{\exp(o_c)}{\sum_{k=1}^{C} \exp(o_k)} \tag{6.19}$$

Si la sortie o_c est la plus grande des valeurs o_k $(1 \leq k \leq C)$, alors $\exp(o_c)$ est aussi la plus grande des valeurs de $\exp(o_k)$ $(1 \leq k \leq C)$ (car l'exponentielle est une fonction croissante), mais toutes ces valeurs sont positives et les grandes valeurs sont fortement favorisées. La fonction "softmax" calcule alors la proportion représentée par la sortie $\exp(o_c)$ et toutes les valeurs sont entre 0 et 1. Si o_c est très majoritaire, alors $\sigma(o_1, ..., o_C)_c \simeq 1$

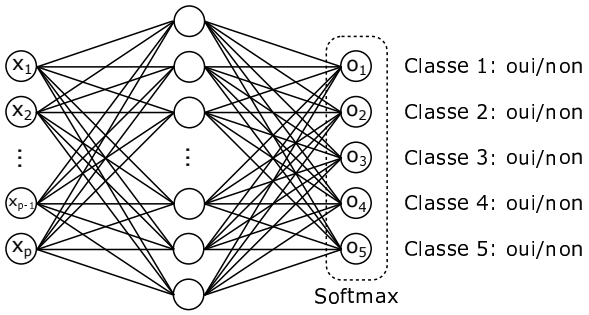


FIGURE 6.11 – Forme générale d'un réseau de neurones pour la classification multiclasse (ici 5 classes) avec activation softmax de la couche de sortie.

et $\sigma(o_1, ..., o_C)_{k \neq c} \simeq 0$. Cette fonction "softmax" n'est en fait qu'un version régulière (dérivable) de la fonction "arg max".

6.6 Méthodes ensemblistes

Certaines méthodes de classification sont des classificateurs faibles, c'est-à-dire que leur taux d'erreur peut être élevé. Si on construit un arbre de décision sur un ensemble de données, il contiendra des erreurs de prédiction. En raison de la multiplicité des arbres possibles (Figure 6.8), si on construit à nouveau un arbre sur ces mêmes données, il pourra avoir une autre structure et induira aussi certaines erreurs, mais pas nécessairement les mêmes que le premier arbre. Si on construit de cette manière plusieurs arbres et qu'on leur présente une nouvelle donnée étiquetée, certains prédiront la bonne étiquette et d'autres se tromperont, mais en faisant la moyenne des réponses, certaines erreurs (positives et négatives) se compenseront et cette moyenne sera finalement une bonne estimation.

Ce principe, sur lequel se basent les méthodes ensemblistes, cache en fait un principe statistique appelé "sagesse de foules". Si on demande à une personne combien vaut $\sqrt{5885476}$, il est possible qu'elle réponde soit totalement au hasard, soit un ordre de grandeur potentiellement cohérent. En posant la même question à une foule de personnes, la moyenne obtenue sera proche de la bonne réponse : 2426.

Les méthodes ensemblistes construisent donc une large population de classificateurs faibles sur un même ensemble de données, et considèrent leur réponse moyenne pour prendre une décision. Leur construction doit néanmoins satisfaire 3 conditions pour que le résultat soit efficace :

- la diversité des classificateurs : ils doivent couvrir une large gamme d'hyperparamètres.
- l'indépendance des classificateurs : ils doivent être indépendants, ce qui est partiellement assuré par la diversité des hyperparamètres. Néanmoins, cette indépendance est généralement renforcée par les données utilisées pour leur entrainement : un sous-ensemble aléatoire du jeu complet de données est souvent utilisé.
- la décentralisation de la décision : les prédictions nouvelles proposées par ces classificateurs doivent être recombinées (vote à la majorité, moyenne) sans filtrage des réponses ni présélection.

6.6.1 Méthodes parallèles : le bagging

Principe du bagging

Le bagging est une méthode de construction et entrainement parallèle d'une famille de classificateurs faibles. Les différentes étapes sont :

- A partir du jeu de n données complet $\{x^{(i)}; y^{(i)}\}$, on construit B jeux de n données par la méthode Bootstrap (voir section 3.4.2).
- Les B classificateurs sont entrainés en parallèle sur chacun des B jeux de données secondaires.
- Chaque modèle est ensuite exploité pour obtenir une prédiction sur des données nouvelles.
- La décision de classification est enfin prise par vote à la majorité en fonction des prédictions des B modèles. Cette logique de bagging peut aussi être utilisée pour une régression auquel cas la prédiction de valeur est obtenue par prise de moyennes sur les prédictions des B modèles.

Méthode des forêts aléatoires

La méthode des **Forêts aléatoires** exploite ce principe de bagging : plusieurs arbres de décision sont construits de manière aléatoire sur des jeux de données différents, obtenus par bootstrap des données originales.

Le caractère aléatoire réside ici dans le choix des variables discriminantes retenues pour la construction des tests conditionnels. Tous ces arbres sont ensuite exploités comme indiqué ci-dessus.

6.6.2 Méthodes séquentielles : le boosting

Principe du boosting

La méthode de bagging par création parallèle de modèles faibles repose sur l'hypothèse que ces mauvais modèles feront des erreurs différentes que l'on parviendra à compenser par la prise de moyenne finale. Au contraire, le boosting cherche à se concentrer à chaque étape sur les erreurs de l'étape précédente, afin de proposer une nouvelle version du modèle qui corrige ces erreurs.

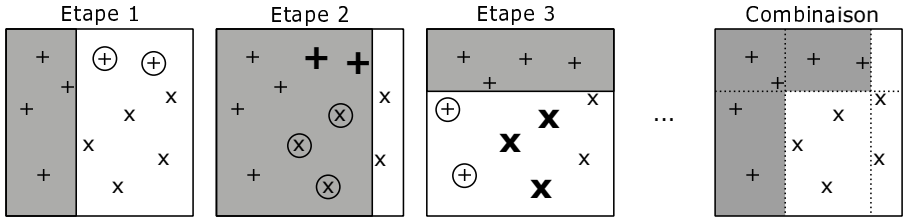


FIGURE 6.12 – Illustration de la logique d'une méthode de boosting.

La Figure 6.12 illustre le principe du boosting sur des données binaires en 2 dimensions. La première étape propose un classificateur qui échoue à représenter deux données +(symboles entourés). La pondération de ces données est augmentée et un nouveau classificateur est entrainé : il parvient à les classer, mais effectue trois nouvelles erreurs sur des données ×. Leur pondération est augmentée et le troisième classificateur parvient à les classer, mais au prix de deux nouvelles erreurs sur des données étiquetées +. Après un nombre donné d'étapes, le classificateur final est construit par pondération des classificateurs intermédiaires.

Adaboost

De nombreuses variantes de la logique de boosting ont été proposées. L'une des premières versions, nommée Adaboost (*Adaptive Boosting*), est encore couramment utilisée dans les environnements de programmation. Ses principales étapes sont :

1. Les poids des n données du jeu complet sont attribués de manière uniforme : $p_{i,m=1} = \frac{1}{n}$ $(1 \leq i \leq n)$.
2. Pour un nombre M d'étapes prédéfini : $1 \leq m \leq M$
   — Entrainer le modèle de classification $f_m(x)$ sur les données pondérées par les poids $p_{i,m}$,
   — Calculer l'erreur pondérée de ce modèle :

$$\epsilon_m = \sum_{i=1}^{n} p_{i,m} \delta\big(f_m(x^{(i)}, y^{((i)}\big)\big) \tag{6.20}$$

   — Calculer la confiance associée :

$$\alpha_m = \frac{1}{2} \log \frac{1 - \epsilon_m}{\epsilon_m} \tag{6.21}$$

   Ce paramètre de confiance est d'autant plus grand que l'erreur ϵ_m est faible. Ce paramètre peut aussi être multiplié par un facteur de relaxation λ (taux d'apprentissage) pour ralentir l'adaptation du modèle.
   — Si l'erreur ϵ_m est inférieure à 0.5, les poids des données mal classées sont multipliés par $\exp(\alpha_m)$. Sinon, l'algorithme est arrêté ou les poids sont réinitialisés.
   — Normaliser les poids pour que leur somme soit égale à 1.
3. La fonction de décision finale est une somme des modèles intermédiaires pondérés par leur confiance :

$$y = \sum_{m=1}^{M} \alpha_m f_m(x) \tag{6.22}$$

Cette version de l'algorithme est la variante "Discrete Adaboost" pour des données discrètes de classification. Il existe une variante "Real Adaboost" pour des données continues en régression (Tufféry, 2019).

6.7 Autres méthodes

6.7.1 Cartes auto-adaptatives, réseaux de Kohonen

Les cartes auto-adaptatives utilisées en partitionnement permettent également de procéder à la classification de nouvelles données. Leur principe est détaillé en section 5.2.

6.7.2 Régression multilinéaire

La régression multilinéaire peut également être exploitée pour réaliser une classification de données. Les caractéristiques de la régression multilinéaire sont rappelées en section 2.2. Cette méthode permet de construire un modèle sous la forme :

$$y_{mod} = f(x; a) \tag{6.23}$$

où la fonction f dépend des composantes x_j des données $x^{(i)}$, éventuellement de façon non-linéaire, mais dépend linéairement des paramètres inconnus a.

Comme l'estimation par une telle fonction, y_{mod}, appartient à $\mathbb{R}$, elle ne peut pas être directement comparée aux étiquettes des données à classer, qu'il s'agisse de données binaires ou multiclasses. Il est donc nécessaire de définir une fonction de seuillage pour traduire la variable continue y_{mod} en une variable discrète compatible avec les étiquettes $y^{(i)}$. Une fonction de seuillage possible pour des étiquettes binaires est :

$$\begin{cases} y^{(i)}_{mod} = 0 & \text{si} \quad y_{mod,i} > 0 \\ y^{(i)}_{mod} = 1 & \text{si} \quad y_{mod,i} \leq 0 \end{cases} \tag{6.24}$$

La difficulté avec cette méthode est le choix de la fonction $f(x; a)$ qui doit être capable de représenter les frontières entre les données d'étiquettes différentes. Des expressions faisant intervenir un grand nombre de données peuvent être exploitées. Pour des frontières fortement non-linéaires, ou des données multimodales, la classification par réseau de neurones peut sembler plus appropriée.

6.7.3 Inférence et prédiction bayesienne

Même si l'essentiel des méthodes d'apprentissage automatique font appel à des méthodes statistiques pour décrire la qualité des prédictions obtenues, beaucoup d'entre elles semblent reposer sur des algorithmes et des logiques plus intuitives que mathématiques : la méthode des plus proches voisins et les arbres de décision ne reposent pas sur des concepts de probabilités et de distributions. Il existe néanmoins plusieurs méthodes qui reposent sur des concepts fondamentaux de statistiques : les méthodes d'inférence bayesienne en font partie, ainsi que leurs dérivées telles que l'analyse discriminante.

Pour comprendre leur importance, il faut revenir à la question qu'on se pose lorsqu'on fait appel aux méthodes automatiques de classification : si je connais les variables x_j d'une nouvelle donnée, quelle est son étiquette ? Puisque plusieurs étiquettes sont potentiellement possible, il faudrait plutôt dire "quelle est la probabilité que son étiquette soit 0 (ou 1) ?". En termes statistiques, on cherche donc à estimer :

$$P(y = 0 \mid x)? \tag{6.25}$$

qui se lit "Quelle est le probalité que y soit égal à 0 connaissant x ?".

Puisque x est le point représentant une donnée dans un espace, p.e. $\mathbb{R}^p$, x peut lui-même être associé à une distribution de probabilité. De même, puisque l'étiquette y peut prendre plusieurs valeurs, il existe aussi une distribution de probabilité des différentes valeurs possibles de y. Ces différentes lois de probabilités sont reliées par une théorème fondamental de statistiques, le **théorème de Bayes**, qui s'écrit :

$$P(y = c \mid x) = \frac{P(y = c)P(x \mid y = c)}{P(x)} \tag{6.26}$$

où :

- $P(y = c)$ désigne la *distribution a priori* des étiquettes, avant même d'avoir observé les données. C'est la distribution originelle des étiquettes à partir de laquelle les étiquettes observées dans les données ont été échantillonnées.
- $P(x \mid y = c)$ désigne la vraisemblance selon laquelle on peut observer la réalisation x sachant que sa classe est c.
- $P(x)$ est la distribution marginale que x soit observée indépendamment de la classe des étiquettes.

Par manque de temps, les méthodes d'inférience bayesienne ne seront pas abordées dans cette option et ne seront donc pas développées dans ce document. Le lecteur intéressé pourra consulter Aggarwal (2015), Goodfellow (2018) et Azencott (2019).

6.8 Applications en Génie des Procédés

6.8.1 Matériaux

Rothenberg (2008) passe en revue les différentes méthodes utilisées en catalyse, notamment les arbres de décision.

Simon et al. (2015) exploitent les arbres de décision et les forêts aléatoires pour identifier les meilleurs matériaux pour séparer du xénon et du krypton par adsorption sélective. Ils entrainent 1000 arbres de décision à partir de 15 000 exemples de matériaux extraits d'une base de 670 000 formulations possibles.

Fernandez et Barnard (2016) combinent une réduction de dimension par ACP avec un partitionnement par K-moyennes et plusieurs méthodes de classification (SVM, MLR, kNN, ANN, arbres de décision) pour identifier les matériaux MOFs les plus appropriés parmi une banque de 82 000 formulations afin d'optimiser les capacités d'adsorption de CO_2 et N_2.

Aghaji et al. (2016) utilisent des méthodes SVM et des arbres de décision pour choisir, dans une base de 320 000 formulations possibles, des matériaux MOFs présentant les plus grandes capacités d'adsorption et les meilleures sélectivités pour la purification de méthane mélangé à du CO_2. Les méthodes de classification permettent de focaliser le choix des formulations pour réduire le nombre de simulations moléculaires à réaliser.

Attarian Shandiz et Gauvin (2016) comparent différentes méthodes de classification (SVM, kNN, LDA, abres, etc.), appliquées à une base de 339 formulations, pour identifier et prédire les formes cristallines de matériaux cathodiques pour des batteries lithium-ion.

Fernandez et al. (2017) utilise des arbres de décision pour classifier des nanoparticules de platine pour la catalyse en fonction de leurs propriétés de structures cristallographiques en vue d'améliorer leur efficacité de conversion.

6.8.2 Détection et diagnostic de fautes et dérives

Venkatasubramanian et al. (1989) sont parmis les premiers à étudier les caractéristiques de réseaux de neurones multicouches (Deep Learning) pour la classification de données de procédés en vue de la détection d'anomalies de fonctionnement. Ils comparent plusieurs réseau (perceptrons monocouches de 18-5-13 à 18-27-13 et multicouches jusqu'à 18-27-27-13) pour relier les 18 mesures issues d'un procédé aux 13 types de défauts observés.

Hoskins et al. (1991) testent un réseau de neurones de Kohonen multicouches 418-40-30-20 reliant les 418 capteurs d'un procédé de synthèse aux 19 types d'anomalies que ce procédé peut subir.

Fan et al. (1993) développent un réseau de neurones 21-13-5 modifié, qui inclut des fonctions sinusoïdales afin d'élargir son potentiel de généralisation grâce à des non-linéarités, et l'appliquent à un procédé d'aromatisation heptane-toluène pour identifier les causes de 5 types d'anomalies.

6.8.3 Sélection d'équipements, synthèse et design de procédés

Voyant et al. (2017) décrit les méthodes de classification utilisées pour la prédiction du rayonnement solaire dans les procédés impliquant du solaire photovoltaique ou thermique : Analyse discriminante, inférence bayesienne naïve, etc.

Chapitre 7

Réseaux de neurones

Conceptualisée dans les années 50, l'idée de neurone artificiel visait initialement à comprendre et simuler informatiquement le fonctionnement des neurones du cerveau humain. Très rapidement, ses propriétés mathématiques ont été exploitées pour d'autres objectifs, mais son utilisation a vite été limitée par les puissances de calcul des calculateurs de l'époque. L'intérêt pour les neurones artificiels a redémarré dans les années 80 grâce à l'augmentation des puissances de calcul qui rendait alors possible l'estimation des nombreux paramètres que contient un réseau de neurones. Néanmoins, la communauté s'est à nouveau retrouvée bloquée par des problèmes méthodologiques, qui n'ont trouvé de réponse efficace que dans les années 90. Depuis ce dernier hiver de l'intelligence artificielle, leur développement est devenu exponentiel et beaucoup des méthodes les plus prometteuses sont construites sur la notion de réseau de neurones.

7.1 Neurone formel

Aussi parfois appelé "Perceptron", le neurone formel est l'entité fondamentale qui constitue la base des réseaux de neurones. On l'appelle "neurone formel" (ou neurone artificiel) par opposition au neurone naturel biologique.

7.1.1 Fonction de transfert

Un neurone est une forme de fonction mathématique élémentaire (Figure 7.1), qui reçoit des variables, les transforme et renvoit un résultat :

- les entrées sont ici notées x_i $(1 \leq i \leq N)$. Il s'agit de variables scalaires, réelles ou entières. Il pourrait s'agir d'autres types de variables, mais elles doivent pouvoir être multipliées et additionnées.
- chacune de ces entrées est multipliée par un poids synaptique (par analogie aux synapses d'un neurone biologique), w_i, réel.
- le neurone fait tout d'abord la somme des entrées pondérées par leurs poids synaptiques respectives,
- le neurone ajoute ensuite un biais, b, réel lui aussi,
- le neurone applique ensuite à cette somme une fonction d'activation, $f(act)$, généralement non-linéaire dont les principales sont décrites ci-dessous. Historiquement, cette fonction d'activation avait pour but de saturer le signal de sortie à 0 ou 1, pour simuler le fait que ce neurone artificiel est dans un état passif ou actif.
- le signal est ensuite renvoyé vers une sortie de réponse ou un autre neurone.

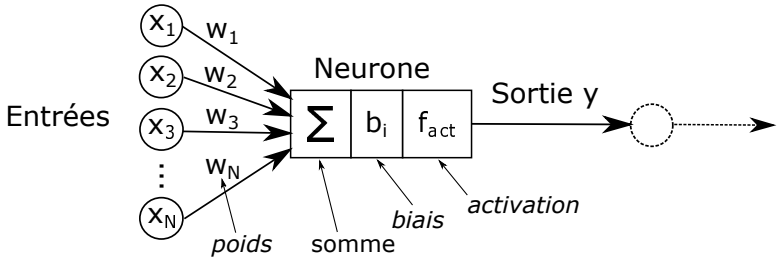


FIGURE 7.1 – Représentation d'un neurone formel.

La sortie de ce neurone isolé peut donc s'exprimer par une fonction mathématique, appelée "fonction de

transfert" (même dénomination, mais concept différent de la fonction de transfert utilisée en commande des procédés), qui s'exprime par :

$$y(x_i, w_i, b) = f_{act}\left(\sum_{i=1}^{N} w_i x_i + b\right) \tag{7.1}$$

En général, les entrées x_i sont des grandeurs expérimentales ou extraites de données diverses. Les poids synaptiques x_i et le biais b sont les $N+1$ paramètres du neurone, qu'il faudra fixer, caler ou identifier par des méthodes particulières en fonction des objectifs visés. La sortie y calculée par ce neurone peut être comparée à un résultat expérimental ou utilisée pour d'autres opérations. Enfin, la fonction d'activation est un hyperparamètre du neurone : il en existe plusieurs qui ont différentes propriétés et il est du ressort de l'ingénieur de choisir la plus adaptée à un problème donné (par expertise ou par essais).

7.1.2 Fonction d'activation

Le Tableau 7.1 présente les principales fonctions d'activation utilisées dans les neurones artificiels, ainsi que leurs dénominations dans l'environnement Matlab. La Figure 7.2 présente la visualisation des ces fonctions.

Fonction d'activation	Expression $y = f_{act}(x)$	Implémentation Matlab
A seuil, Heaviside	$\begin{cases} y = 0 & si \quad x < 0 \\ y = 1 & si \quad x \geq 0 \end{cases}$	`hardlim`
Linéaire	$y = x$	`purelin`
ReLU	$\begin{cases} y = 0 & si \quad x < 0 \\ y = x & si \quad x \geq 0 \end{cases}$	`poslin`
Leaky ReLU	$\begin{cases} y = 0.01x & si \quad x < 0 \\ y = x & si \quad x \geq 0 \end{cases}$	
Logistique, Sigmoïde	$y = \frac{1}{1+e^{-x}}$	`logsig`
Tangente hyperbolique	$y = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	`tansig`
Radiale	$y = exp(-x^2)$	`radbas`
Softmax	$y(z_i) = f_{Softmax}(z_i) \frac{e^{z_i}}{\sum_{k=1}^{N} e^{z_k}}$	`softmax`

TABLE 7.1 – Principales fonctions d'activation de neurones.

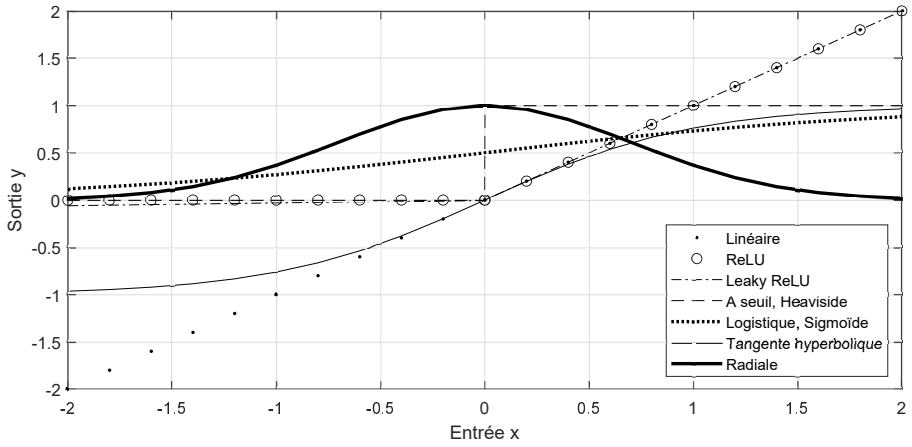


FIGURE 7.2 – Comparaison des principales fonction d'activation d'un neurone.

Historiquement, c'est la fonction à seuil de Heaviside qui était utilisée pour qu'un neurone s'allume ($y = 1$) ou s'éteigne ($y = 0$) en fonction des signaux qu'il recevait. Si cette utilisation semblait logique, cette fonction présente l'inconvénient majeur de posséder une dérivée nulle en tout point : la sortie y est donc très peu sensible aux valeurs des poids et du biais. Or ces paramètres ont besoin d'être optimisés dans certains cas, ou au moins entraînés, et ce manque de sensibilité rend toute optimisation impossible.

La fonction linéaire est sans effet d'activation puisqu'elle ne modifie pas le signal calculé à partir des poids et biais. C'est une des rares fonctions d'activation non-bornée. Elle est généralement plutôt utilisée en toute fin des réseaux de neurones sur la couches de sorties.

Comme la fonction linéaire ne fait rien, sa petite soeur "ReLU" (Rectified Linear Unit) a été construite pour introduire un effet de filtre : seule la partie positive est conservée, la partie négative étant saturée à 0. Elle est très utilisée en traitement d'images. Mais la fonction ReLU possède également une dérivée nulle pour toutes les entrées négatives, ce qui pose, comme pour la fonction de Heaviside, un problème pour l'optimisation des poids et biais. La fonction "Leaky ReLU" (ReLU "percée") a donc été construite : une très faible pente est imposée dans la partie négative.

Les deux fonctions saturées (bornées asymptotiquement) mais ne présentant pas de dérivées nulles sont la sigmoïde (aussi appelée fonction logistique) et la tangente hyperbolique. Elles possèdent deux asymptotes horizontales, à 0 et 1 pour la sigmoïde, et à -1 et 1 pour la tangente hyperbolique. Ces bornes ne sont pas généralement le premier argument pour choisir l'une ou l'autre, parce que la forme de la courbe a plus d'impact que ses valeurs (de toute façon ajustables par composition linéaire). Par contre, un argument de poids donne l'avantage à la sigmoïde et à la tangente hyperbolique pour les méthode d'entraînement par rétropropagation car leurs dérivées sont pratiques à utiliser dans la règle de la chaine pour les dérivées enchaînées :

$$f'_{sigmoide}(x) = f_{sigmoide}(x)\Big(1 - f_{sigmoide}(x)\Big) \tag{7.2}$$

$$f'_{tanh}(x) = 1 - f^2_{tanh}(x) \tag{7.3}$$

L'autre intérêt de cette fonction sigmoïde est son analogie à une distribution cumulée de probabilité : elle permet donc d'associer à la frontière entre le domaine où la sortie vaut 0 et celui où elle vaut 1, une largeur correspondant à l'écart-type de cette distribution de probabilité. Cette forme établit aussi des passerelles avec les méthodes statistiques, notamment les statistiques bayésiennes, très utilisées en fouille de données et en intelligence artificielle, mais non traitées ici.

La fonction d'activation radiale fait également le lien avec les statistiques de par sa forme gaussienne. C'est une des rares fonctions d'activation possédant la même limite pour les valeurs de x très positives et très négatives.

La dernière fonction d'activation présentée dans le Tableau 7.1 est la fonction "Softmax". Elle est particulière puisqu'elle ne s'applique pas en sortie d'un neurone unique, mais elle travaille sur les sorties de plusieurs neurones. C'est une sigmoïde généralisée à plusieurs signaux. Plus précisément, pour différentes variables z_i sortant de plusieurs neurones, il existe un minimum et un maximum des z_i, z_{min} et z_{max}. La fonction $f_{Softmax}(z_i)$ renvoit une valeur d'autant plus proche de 1 que z_i est proche de z_{max} et d'autant plus proche de 0 que z_i est proche de z_{min}. De la même manière que la fonction sigmoïde est une version adoucie de la fonction de Heaviside, la fonction Softmax est une version adoucie d'une fonction permettant d'identifier le maximum parmi un ensemble de valeurs. Si les N valeurs de z_i sont égales, $f_{Softmax}(z_i)$ vaut $\frac{1}{N}$.

7.1.3 Types de sorties d'un neurone à 1 entrée

Même si l'expression de la sortie y d'un neurone artificiel n'est pas très compliquée, il n'est pas évident de voir le type de fonction qu'il renvoie. La Figure 7.3 présente les sorties de différents neurones basés sur les principales fonctions d'activation. Dans chaque cas, on ne considère qu'un seul neurone à une seule entrée x, un poids w_1 et un biais b. Sur chaque figure, on trace en fonction de x la réponse $y(x, w_1, b)$: les courbes se distinguent par des valeurs différentes de w_1 et b, fixées aléatoirement entre -10 et 10.

Pour une activation linéaire, quels que soient w_1 et b, la sortie y est une droite dont la pente et l'ordonnée à l'origine dépendent de w_1 et b : le neurone unique à activation linéaire reste linéaire. On peut remarquer que les deux paramètres w_1 et b ont un impact sur la réponse puisque la pente et l'ordonnée à l'origine sont impactées. On retrouve cette linéarité apparente avec les fonctions ReLU et "Leaky ReLU", mais qui sont effectivement saturée ou partiellement saturée, puisqu'il n'y a pas ou très peu de valeurs négatives.

L'activation à seuil de Heaviside semble donner des droites très verticales, mais il s'agit d'un artefact graphique : toutes les courbes sont en fait des marches entre 0 et 1 (ou entre 1 et 0), la seule partie visible ici étant l'abscisse x où la fonction change de valeur, x_{seuil}. La modification de w_1 et b, modifie donc la valeur de x_{seuil} et le sens de la marche montante ou descendante : les deux paramètres influencent les deux caractéristiques de la réponse.

Les activations en sigmoïde ou en tangente hyperbolique fournissent des réponses qui se ressemblent beaucoup : des courbes en "S" plus ou moins aplaties, croissantes ou décroissantes, et passant par leur valeur médiane (0.5 pour la sigmoïde et 0 pour la tangente) à des abscisses différentes. On remarque ici que les deux paramètres w_1 et b impactent 3 grandeurs caractéristiques de la réponse. On peut également remarquer que certaines courbes ressemblent beaucoup soit à des droites, soit à des marches : les sigmoïdes ont donc le potentiel pour représenter à la fois des courbes, des droites ou des fonctions à seuil : cette flexibilité est une de leurs forces qui explique leur utilisation massive dans les réseaux de neurones.

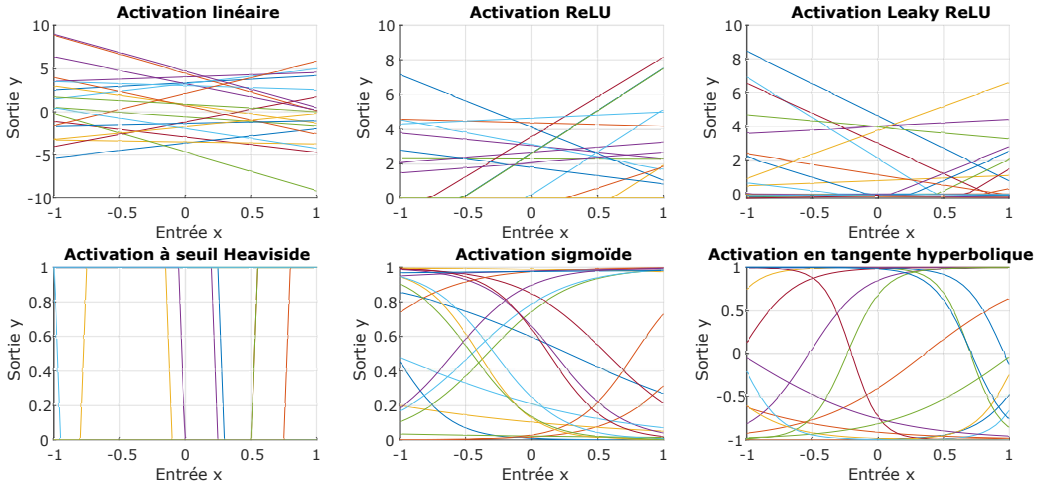


FIGURE 7.3 – Types de réponses que renvoie un neurone en fonction de son entrée x et de son type d'activation pour des poids et biais aléatoires.

7.1.4 Entraînement d'un neurone formel

Une fois la structure d'un neurone (ou d'un réseau) fixée, l'objectif est de lui permettre de prédire une sortie particulière en fonction des entrées qu'on lui donne. La Figure 7.4 présente un neurone avec N entrées et une sortie, ainsi qu'un tableau de données correspondant à chaque expérience réalisée. Chaque ligne de ce tableau correspond à une expérience avec ses conditions opératoires (facteurs x_1 à x_N) et ses résultats (réponse y_e, appelée "étiquette").

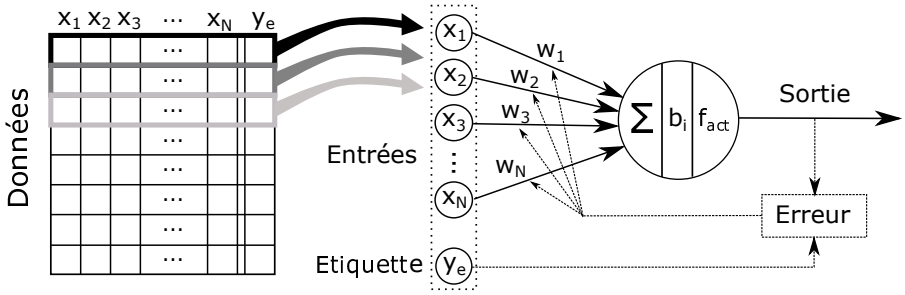


FIGURE 7.4 – Logique de rétropropagation.

Pour que le neurone modélise parfaitement une expérience, il faut, lorsqu'on lui présente en entrée les facteurs x_1 à x_N de l'expérience, qu'il soit capable de calculer une sortie qui doit être égale à l'étiquette y_e de cette expérience, ou tout du moins aussi proche que possible. Et cela doit être vérifié également pour toutes les expériences du tableau de données. Or, lorsqu'on construit le réseau de neurones, on n'a encore aucune idée des valeurs idéales qu'il faudrait attribuer aux poids w_i et au biais b, ni de la fonction d'activation qu'il faut choisir : tous ces paramètres doivent être calés, ou identifiés ou optimisés, lors de la phase d'entraînement.

En apprentissage automatique, on parle d'entraînement plutôt que d'optimisation ou d'identification car on ne va pas nécessairement cherche à atteindre l'optimum exactement, mais plutôt un compromis entre une bonne qualité de prédiction et une bonne capacité de généralisation. Néanmoins, la logique générale est la même qu'une optimisation qui cherche à trouver les poids et biais qui minimisent l'écart modèle-expérience sur l'ensemble des données disponibles.

Pour réaliser cet entraînement, l'idée consiste à présenter au neurone les différentes expériences disponibles de manière séquentielle et à réactualiser les poids régulièrement en fonction de la réussite ou de l'échec des prédictions. Pour un ensemble de poids et biais donnés (par exemple, par initialisation aléatoire), on applique en entrée les facteurs d'une expérience. On compare l'étiquette de cette expérience avec la sortie prédite par le neurone/réseau pour calculer l'erreur d'apprentissage sur cette expérience. Puis, en fonction de cette erreur, on modifie légèrement les poids dans le but de réduire l'erreur. Ce retour d'information a donné lieu à la dénomination "Rétropropagation" qui est une des méthodes classiques d'entraînement et qui consiste à propager en sens inverse des sorties vers les entrées les termes correctifs qui vont permettre d'améliorer séquentiellement les paramètres du réseau.

7.1.5 Fonctions d'erreur, fonctions de coût

Pour corriger les paramètres du réseau par rétropropagation (ou par algorithmes d'optimisation), il est nécessaire de quantifier l'écart modèle-expérience. Il existe pour cela plusieurs fonctions d'erreurs (aussi appelées "fonctions de coût"), dont les plus courantes sont présentées dans le Tableau 7.2.

Fonction d'activation	Expression
Erreur Quadratique Moyenne (MSE)	$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y_i^{net} - y_i^{exp}\right)^2$
Root-mean Squared Error (RMSE)	$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(y_i^{net} - y_i^{exp}\right)^2}$
Root-mean Squared Log Error (RMSLE)	$RMSLE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\log(y_i^{net}+1) - \log(y_i^{exp}+1)\right)^2}$
Erreur Absolue Moyenne (MAE)	$MAE = \frac{1}{N}\sum_{i=1}^{N}\left\lvert y_i^{net} - y_i^{exp}\right\rvert$
Erreur Relative Moyenne (MRE)	$MRE = \frac{1}{N}\sum_{i=1}^{N}\frac{y_i^{net}-y_i^{exp}}{y_i^{exp}}$
Ecart Absolu Moyen (AAD)	$AAD_{\%} = \frac{100}{N}\sum_{i=1}^{N}\left\lvert\frac{y_i^{net}-y_i^{exp}}{y_i^{exp}}\right\rvert$
Erreur Absolue Relative Maximale (MARE)	$MARE = 100 Max_i\left(\left\lvert\frac{y_i^{net}-y_i^{exp}}{y_i^{exp}}\right\rvert\right)$

TABLE 7.2 – Principales fonctions d'erreur utilisées pour l'entraînement de réseaux de neurones. N désigne le nombre total de données prises en compte. y_i^{exp} représente l'étiquette expérimentale de l'expérience i et y_i^{net} la prédiction fournie par le réseau pour cette expérience.

Pour pourvoir minimiser l'écart et le rendre aussi proche que possible de 0, il est important que la fonction d'erreur soit toujours positive, d'où la présence d'expressions quadratiques ou en valeurs absolues. Les fonctions quadratiques sont les plus utilisées car elles sont dérivables en tout point et possèdent une dérivée nulle au minimum, alors que les fonctions en valeurs absolues présentent des points anguleux et sont plus utilisées en post-traitement pour le calcul d'écarts moyens. Les expressions "relatives" où l'écart est divisé par la valeur expérimentale sont particulièrement utiles lorsque le réseau de neurones possède plusieurs sorties dont les ordres de grandeurs sont différents : sans cette "normalisation", les sorties de plus grande amplitude deviennent majoritaires dans le calcul d'erreur et le réseau apprend mal à représenter les sorties de plus faible amplitude.

7.2 Perceptron monocouche

Comme un neurone unique est trop limité en termes de type de modèles atteignables (Figure 7.3), les neurones sont agencés en parallèle pour augmenter les degrés de liberté des fonctions qu'ils peuvent représenter. Ces neurones parallèles sont associés en couches entre la couche d'entrée et la couche de sortie.

7.2.1 Expression

Le perceptron monocouche ne contient qu'une seule couche de neurones (Figure 7.5). On l'appelle "couche cachée" car l'intérieur du réseau est assimilé à une boite noire : on ne "voit" que ce qui rentre (couche d'entrées) et ce qui sort (couche de sorties), tout le reste est "caché". Pour la suite, on notera N_i le nombre d'entrées (inputs), N_h le nombre de neurones dans la couche cachée (hidden) et N_o le nombre de sortie (outputs). L'indice i correspond à une entrée, j à un neurone caché et k à une sortie. Sous forme compacte, un tel réseau s'écrit $N_i - N_h - N_o$.

La Figure 7.5 présente deux représentations du même objet. La forme la plus conventionnelle est celle de gauche : les N_i entrées envoient leurs valeurs aux N_h neurones de la couche cachée. Chaque neurone fait la somme pondérée de ses entrées, ajoute son biais b_j, applique une fonction d'activation et envoie sa réponse à toutes les sorties. Chaque neurone de la couche cachée fait subir à ses entrées la même transformation que présentée sur la Figure 7.1. Dans la couche de sortie, chaque neurone fait pareil : somme pondérée des entrées, ajout d'un biais, application d'une fonction d'activation. Les poids $w_{i,j}$ et $w_{j,k}$ peuvent être représentés sous formes de matrices, $W_{ij}(N_i \times N_h)$ et $W_{jk}(N_h \times N_o)$.

Chaque sortie de ce réseau monocouche peut être exprimée en fonction des entrées telle que :

$$y_k = f_{act,o}\left[\sum_{j=1}^{N_h} w_{j,k} f_{act,h}\left(\sum_{i=1}^{N_i} w_{i,j}x_i + b_j\right) + b_k\right] \tag{7.4}$$

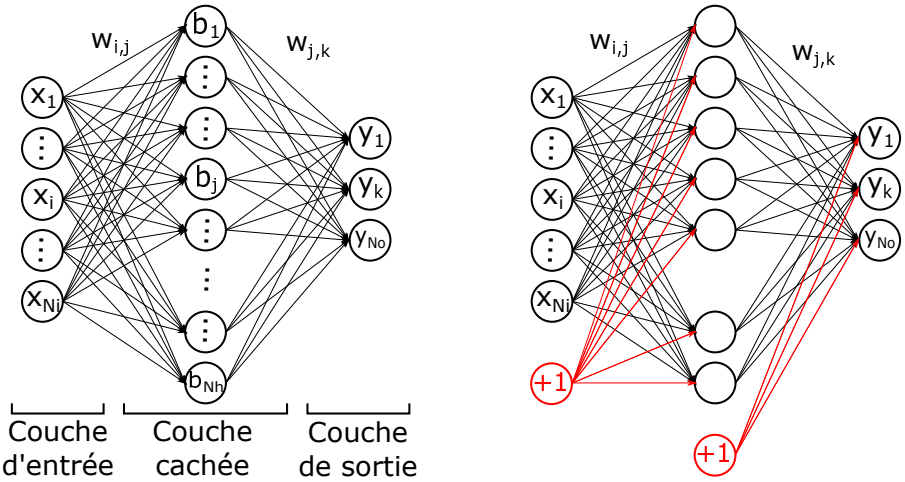


FIGURE 7.5 – Structure générique d'un perceptron mono-couche $N_i - N_h - N_o$.

où les fonctions d'activation dans la couche cachée $f_{act,h}$ et dans la couche de sortie $f_{act,o}$ ne sont par forcément les mêmes.

Le schéma à droite de la Figure 7.5 est une représentation alternative de ce même réseau. Plutôt que d'inclure les biais dans les neurones cachés et de sortie, on ajoute des sources "+1" dans la couche d'entrée et dans la (ou les) couches(s) cachée(s). Les neurones ne font que la somme pondérée et l'application de la fonction d'activation : les paramètres b_j et b_k n'existent plus sous ce nom, il sont en fait inclus dans les dernières lignes tableaux de poids, dont les dimensions changent : $W^+_{ij}((N_i + 1) \times N_h)$ et $W^+_{jk}((N_h + 1) \times N_o)$. Ce formalisme ne change rien au potentiel du réseau, mais il simplifie les écritures pour certaines méthodes d'entraînement.

7.2.2 Paramètres et hyper-paramètres

Dans l'expression de la réponse du réseau, on distingue les paramètres et les hyper-paramètres. Les paramètres incluent les poids et les biais. Pour un réseau monocouche $N_i - N_h - N_o$, le nombre total de poids et de biais qu'il faudra entraîner est :

$$N_{param} = N_i.N_h + N_h + N_h.N_o + N_o = N_h(N_i + 1) + N_o(N_h + 1) \tag{7.5}$$

Les hyper-paramètres sont :

— le nombre d'entrées N_i : ce nombre dépend des données disponibles et des facteurs qui ont été testés expérimentalement. Mais il est possible aussi de construire des entrées composites en fonction des facteurs : par exemple, si x_1 et x_2 sont deux facteurs étudiés, rien n'interdit d'utiliser aussi $x_1.x_2$ comme une entrée.

— le nombre de sorties N_o : ce nombre dépend des données disponibles.

— le nombre de neurones dans la couche cachée N_h : c'est un hyper-paramètre important, mais difficile à intuiter. Son réglage se fait généralement par essais-erreurs et il existe des heuristiques pour en connaître un ordre de grandeur en fonction des données disponibles (Section 7.6)

— les fonctions d'activation $f_{act,h}$ et $f_{act,o}$: leur choix dépend du problème visé (régression, classification) et des données. Très souvent, on utilise une activation linéaire en sortie et une activation sigmoïde (ou tangente hyperbolique) dans les couches cachées.

Les paramètres sont calés grâce aux méthodes d'entraînement abordées ci-dessous. Il existe des méthode d'entraînement adaptatives qui entraînent à la fois les paramètres et les hyper-paramètres, mais elles ne seront pas abordées dans ce document.

7.2.3 Forme des réponses atteignables

La Figure 7.6 présente les types de réponses $y(x)$ que permet de générer un simple perceptron 1-2-1 (1 entrée x, 2 neurones cachés et 1 sortie y) avec 7 paramètres. Pour les 10 courbes tracées, les paramètres ont été tirés aléatoirement. L'activation est sigmoïde dans la couche cachée et linéaire en sortie.

On retrouve sur ces courbes des formes caractéristiques de la sigmoïde, avec des "marches" : parfois deux marches de même hauteur s'ajoutent l'une à l'autre, parfois ce sont une grande et une petite marche. Parfois, les marches se soustraient l'une l'autre et la courbe ressemble à un pic quasi-gaussien ou asymétrique. On retrouve

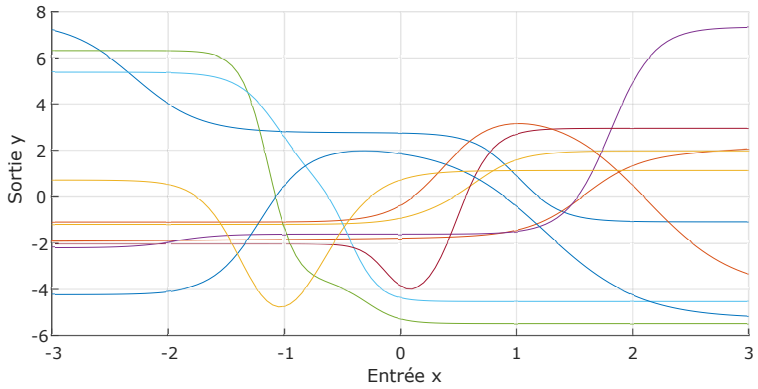


FIGURE 7.6 – Exemples de courbes que permet de modéliser un perceptron à 1 entrée, 2 neurones et 1 sortie avec activation en tangente hyperbolique (4 poids + 3 biais = 7 paramètres ici aléatoires) en fonction de l'entrée x.

des formes connues en Génie des Procédés : courbes de dosages acide-bases, courbes de percées d'une colonne d'adsorption, courbe de rendement d'un système de réactions consécutives en fonction du temps de séjour, distribution de poids moléculaires, ressaut hydraulique, etc. Plus on ajoute de neurones dans la couche cachée, plus les fonctions atteignables peuvent être complexes : en fait, on peut approximer n'importe quelle fonction en intégrant suffisamment de neurones.

7.2.4 Théorème d'approximation universelle

Une propriété importante des réseaux de neurones est leur propriété d'approximation universelle, qui a été démontrée par Cybenko (1989) et qu'on résumer ainsi (Azencott, 2019) :

> *Toute fonction bornée suffisamment régulière peut être approchée uniformément, avec une précision arbitraire, dans un domaine fini de l'espace de ses variables, par un réseau de neurones comportant une couche de neurones cachés en nombre fini, possédant tous la même fonction d'activation, et un neurone de sortie linéaire.*

Cette propriété d'approximation universelle est assez intuitive si on se rappelle les formes de fonctions que permettent de générer un neurone unique (Figure 7.3) et un perceptron à 2 neurones (Figure 7.6) avec une fonction d'activation sigmoïde, et qu'on se permet une analogie avec des Légo (Figure 7.7).

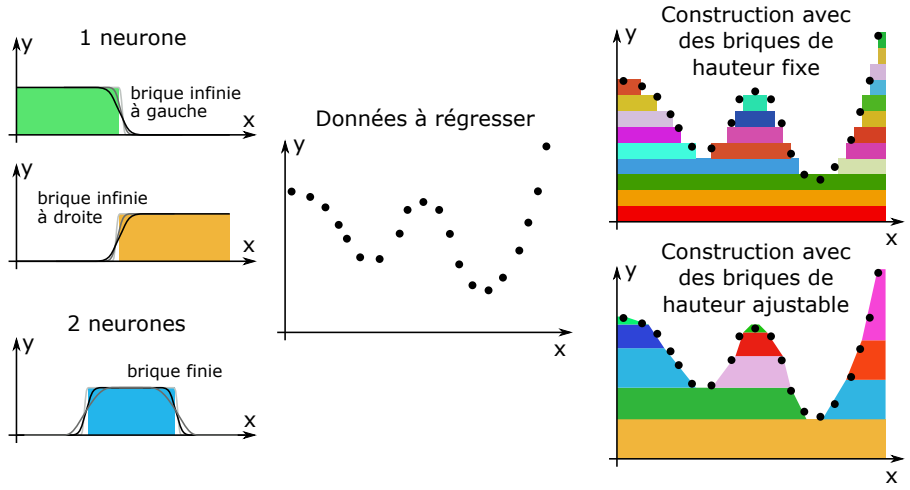


FIGURE 7.7 – Représentation schématique de la capacité d'approximation universelle d'un perceptron monocouche.

Les figures de gauche rappellent les fonctions qu'il est possible de générer avec 1 ou 2 neurones. Avec un neurone, on peut générer une sigmoïde dont l'orientation (montante ou descendante), le point d'inflexion (abscisse de la marche) et la pente (marche raide ou en pente douce) sont réglables grâce au poids et au biais associés à ce neurone. Un neurone permet donc de générer l'équivalent mathématique de briques de Légo infinies

vers la droite ou vers la gauche. Puisque la combinaison de 2 neurones permet d'ajouter les deux marches générées par chacun des deux neurones, il est possible, en choisissant des poids de signes opposés entre la couche cachée et la couche de sorties, de générer des briques de longueur finie, mais dont la hauteur est encore ajustable, ainsi que les pentes des deux côtés.

Il faut aussi se souvenir que ces briques sont générées par les neurones de la couche cachée et que la sortie est une somme pondérée des signaux provenant de la couche cachée : la sortie est donc la somme des neurones. Cette somme est équivalente au fait d'empiler les briques générées par les neurones, comme si on construisait un mur en Légo.

La figure centrale de la Figure 7.7 présente une série de 20 données $y(x)$ qu'on souhaite régresser. Les deux figures à droite représentent deux constructions de cette fonction par empilement de briques :

— dans le premier cas, on n'utilise que des briques de même hauteur dont les côtés pentus ne sont pas ajustables (comme si une fonction d'activation de Heaviside avait été utilisée au lieu d'une sigmoïde). 18 briques infinies et 4 briques finies ont été utilisées, soit un besoin de 26 neurones contraints. L'accord entre la "régression" et les données n'est pas parfait, mais on comprend tout de suite que des briques plus petites auraient permis de mieux approximer la fonction. Et quel que soit le niveau de précision requis, on peut toujours s'approcher de la fonction autant qu'on veut du moment qu'on accepte d'utiliser des briques très petites, donc un grand nombre de neurones.

— dans le second cas, la hauteur et les pentes des côtés sont ajustables. Il n'a fallu ici que 8 briques infinies et 3 briques finies, soit 14 neurones pour approximer la fonction "dans l'épaisseur des points". Il est sûrement possible de faire aussi bien avec moins de neurones.

Cette même logique de construction par empilement peut être appliquée lorsque le nombre de dimensions augmente, par exemple pour une régression en deux dimensions $y(x_1, x_2)$ mais est plus difficile à représenter.

7.3 Réseaux multicouches

Les réseaux multicouches sont construits selon une même logique et avec les mêmes briques élémentaires que les perceptrons monocouches (Figure 7.8). L'unique différence réside dans le nombre de couches cachées, mais le fonctionnement reste identique : chaque neurone d'une couche reçoit une somme pondérée de toutes les sorties des neurones de la couche précédente (couche "dense"). Chaque neurone possède également un biais et une fonction d'activation : sa sortie, non linéaire par rapport à ses entrées, est envoyée à chaque neurone de la couche suivante, ou à la couche de sortie. Le réseau est "à propagation avant", c'est-à-dire que le signal ne se propage que de la gauche vers la droite, il ne revient jamais à contre-courant : les structures particulières, propres au "Deep Learning" modifieront cette structure générale.

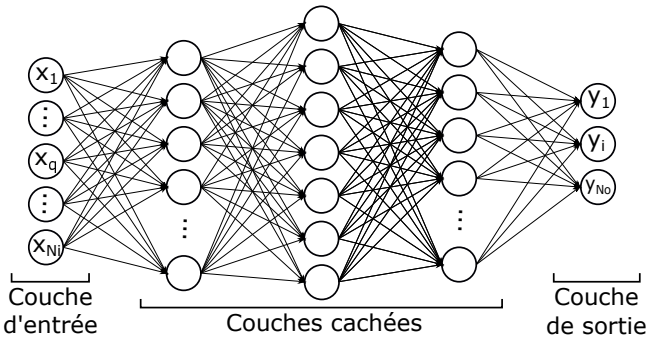


FIGURE 7.8 – Représentation d'un perceptron multicouche.

Cet empilement de couches et de non-linéarités n'est pas nécessairement un avantage. Certes, les perceptrons multicouches satisfont aussi le théorème d'approximation universelle, mais c'était déjà le cas avec un perceptron monocouche : un perceptron monocouche, avec un nombre suffisant de neurones cachés, suffit pour pouvoir approximer toute fonction. L'ajout de couches supplémentaires n'apporte rien au cas de la régression standard.

Cet ajout de couches s'avère, par contre, utile lorsque les données d'entrée possèdent une structure, connue ou à découvrir, comme en analyse d'image où une image peut contenir une sous-structure correspondant à un visage, un chien ou un bateau. C'est cette caractéristique qui fera la force de ces réseaux multicouches qui évolueront vers les réseaux profonds exploités en "Deep Learning". La structure complexe de ces réseaux combine des fonctions de classification, de réduction de dimensionnalité et de régression selon des descripteurs ou des variables latentes.

Si on considère le réseau de la Figure 7.8, on peut exprimer l'une de ses sorties en fonction de ses entrées, des poids et biais de tout le réseau, et des fonctions d'activation des différentes couches. Si on considère un réseau avec 2 couches cachées ($N_i - N_{h_1} - N_{h_2} - N_o$), une activation sigmoïde dans les couches cachées, une activation linéaire dans la couche de sortie, des poids $w^{(1)}_{q,k}$ entre la couche d'entrée (indice q) et la première couche cachée (indice k), des poids $w^{(2)}_{k,j}$ entre les deux couches cachées, et des poids $w^{(3)}_{j,i}$ entre la seconde couche cachée et la sortie, alors la sortie du neurones i de la couche de sortie d'écrit :

$$f_i(x) = \sum_{j=1}^{j=N_{h_2}} w^{(3)}_{j,i} \frac{1}{1+\exp\left(-\sum_{k=1}^{N_{h_1}} w^{(2)}_{k,j} \frac{1}{1+\exp\left(-\sum_{q=1}^{N_i} w^{(1)}_{q,k} x_q\right)}\right)} \tag{7.6}$$

Si l'augmentation du nombre de couches n'apporte pas un gain systématique, il est certain que cela démultiplie le nombre de poids et biais à entrainer/optimiser. L'apprentissage de ces réseaux est donc plus difficile et requiert un nombre de données d'autant plus élevé.

7.4 Calcul et optimisation des poids : entrainement d'un réseau

7.4.1 Limites des méthodes d'optimisation habituelles

Comme indiqué précédemment, pour une structure de réseau et des fonctions d'activation données, l'entraînement consiste globalement à trouver les valeurs optimales des poids et biais qui permettront au réseau de prédire des sorties aussi proches que possible des étiquettes des expériences qu'on lui proposera. Si on note globalement w_j tous les poids et b_k tous les biais à identifier, et w^*_j et b^*_k leurs possibles valeurs optimales, alors ce problème d'entraînement, à partir d'un ensemble de N données expérimentales, peut être écrit sous la forme d'un problème d'optimisation classique :

$$w^*_j, b^*_k = ArgMin_{w_j, b_k} \sum_{i=1}^{N} \left(y^{net}_i - y^{exp}_i\right)^2 \tag{7.7}$$

où y^{net}_i est la prédiction du réseau pour une expérience i. Si la fonction d'activation en tangente hyperbolique est utilisée comme activation de la couche cachée (et l'activation linéaire en sortie), alors, pour un réseau monocouche à 3 entrées x_1, x_2 et x_3, la sortie s'écrit :

$$\begin{aligned} y^{net}_i(x_{1,i}, x_{2,i}, x_{3,i}) =& b_4 + w_{10} \tanh(b_1 + w_1 x_{1,i} + w_2 x_{2,i} + w_3 x_{3,i}) \\ &+ w_{11} \tanh(b_2 + w_4 x_{1,i} + w_5 x_{2,i} + w_6 x_{3,i}) \\ &+ w_{12} \tanh(b_3 + w_7 x_{1,i} + w_8 x_{2,i} + w_9 x_{3,i}) \end{aligned} \tag{7.8}$$

Or, comme la même fonction d'activation est utilisée sur tous les neurones de la couche cachée, cette équation est invariante si on fait une permutation sur les notations des poids et des biais :

$$\{w_1, w_2, w_3, b_4, w_{10}\} \iff \{w_4, w_5, w_6, b_2, w_{11}\} \tag{7.9}$$

Cette invariance par permutation se comprend bien graphiquement avec la Figure 7.9, où le neurone rouge R et le neurone bleu B sont permutés entre la figure de gauche et la figure de droite. Le deux réseaux sont absolument les mêmes structurellement. Donc, si on entraîne ces deux réseaux sur un grand nombre de données et qu'on atteint un optimum pour le premier, alors ce sera aussi un optimum pour le second. En fait, pour ce réseau 3-3-1, il n'existe pas un seul optimum, mais $3! (= 6)$ optima équivalents, correspondant à la même fonction de coût minimale mais résultant des permutations possibles entre les neurones de la couche cachée. Pour une couche cachée à N_h neurones, il existe $N_h!$ permutations : un réseau 3-10-1 à seulement 10 neurones cachés induit donc 3 628 800 solutions optimales équivalentes dans un espace des poids et biais de dimension 51. Dans un réseau multicouches, il existe des invariances par permutations dans toutes les couches successives.

Cette multiplicité par permutations est une difficulté, car le problème contient non seulement des optima globaux multiples, mais aussi des optima locaux multiples car l'invariance par permutation s'applique aussi aux optima locaux. Il est donc très difficile de s'assurer qu'une méthode déterministe converge bien vers un optimum global.

Une autre particularité doit être prise en compte pour l'identification des biais et poids, il s'agit du compromis biais-variance (voir section 2.4) : on ne cherche pas forcément à trouver des poids et des biais qui vont trop bien faire coller le réseau et les mesures expérimentales, car il risquerait alors de reproduire également le bruit associé

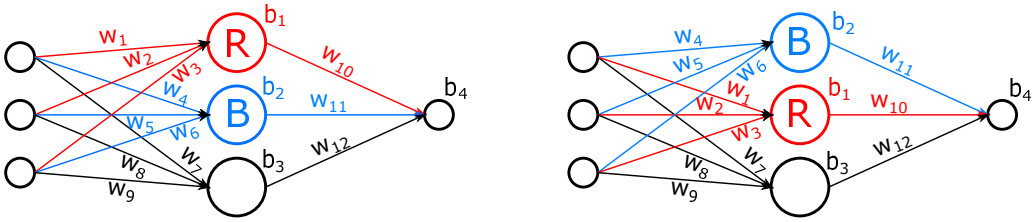


FIGURE 7.9 – Existence de symétries par permutations dans les réseaux de neurones.

aux mesures. Il serait alors incapable de faire des prédictions fiables sur des entrées différentes de celles qu'il a vues pendant son apprentissage : il serait en sur-apprentissage. Les méthodes d'entrainement de réseaux de neurones s'inspirent donc des méthodes d'optimisation, mais possèdent une structure particulière "Entraînement - Validation - Test".

7.4.2 Entraînement - Validation - Test

Comme la grande majorité des méthodes d'apprentissage automatiques, les réseaux de neurones sont soumis à différents types d'erreur : erreur d'entrainement, erreur empirique, erreur de généralisation et erreur de modèle (voir section3.4). L'estimation indépendante de ces erreurs, et surtout de l'erreur de généralisation, nécessite de les faire travailler sur différents jeux de données. L'entrainement d'un réseau de neurones est toujours réalisé sur 2, voire 3, jeux de données issus du jeu de données orginales :

- **Données d'entrainement** : c'est l'ensemble des données utilisées pour figer les poids et les biais du réseau. Si plusieurs méthodes ou jeux d'hyper-paramètres doivent être comparés, tous les modèles sont entrainés sur ce même jeu d'entrainement.
- **Données de validation** : ce sont les données utilisées pour comparer les éventuels jeux d'hyper-paramètres. Le modèle fournissant l'erreur empirique minimale sur ces données est alors sélectionné.
- **Données de test** : ce sont les données utilisées pour estimer l'erreur de généralisation du modèle retenu à l'étape précédente.

Lorsque les hyperparamètres ont préalablement été figés, seules les données d'entrainement et les données de test sont utilisées. Les jeux de données d'entrainement et de test sont souvent générès par partition du jeu complet : 2/3 (ou 3/4) des données sont réservées pour l'entrainement, et 1/3 (ou 1/4) est dédié au test.

Pendant l'entrainement, on suit l'évolution de l'erreur sur les données d'entrainement et sur les données de test en fonction du nombre de cycles de présentation des données ("epoch"). L'entrainement est arrêté lorsque l'erreur de test commence à augmenter après avoir significativement baissé. Cette augmentation de l'erreur de test indique que le réseau entre en phase de surapprentissage : il a tellement bien appris à représenter les données d'entrainement qu'il représente non seulement leurs grandes tendances, mais aussi les incertitudes associées. Il ne parvient donc plus à généraliser les tendances observées et sa prédiction est entâchée de l'incertitude des données originales.

7.4.3 Logique de l'entrainement par rétropropagation

Comme l'a montré la Figure 7.4, l'idée générale de l'entrainement des réseaux de neurones par rétropropagation consiste à (i) présenter une donnée aux entrées du réseau, (ii) propager ces signaux dans le sens "avant" à travers les différentes couches cachées jusqu'à la sortie, (iii) comparer cette sortie prédite à la valeur réelle de l'étiquette de la donnée, et (iv) modifier les poids et biais du réseau en fonction de cet écart. Selon le type de problème (régression, classification, etc.) et le nombre de couches cachées, l'actualisation des poids se fait selon différentes méthodes, mais la logique est généralement la même : les erreurs en sortie sont rétropropagées, depuis la sortie vers l'entrée, pour modifier progressivement chaque couche du réseau.

Les sections ci-dessous présentent quelques cas simples et les grandes lignes du cas général de la rétropropagation dans un réseau multicouche. Le lecteur intéressé est invité à consulter la nombreuse littérature sur le détail de ces méthodes (Azencott, 2019 ; Tufféry, 2019).

Cas d'un perceptron simple pour la classification binaire

On considère ici un perceptron simple utilisé pour la classification binaire, caractérisé par des poids w_j (Figure 7.10). On notera $\{x_j^{(i)}(1 \leq j \leq N_i); y^{(i)}\}$ les données étiquetées d'entrée et $f(x^{(i)})$ la prédiction de l'étiquette de la donnée i par le réseau. La fonction d'erreur dans le neurone de sortie peut être une fonction à

seuil :

$$L\Big(f(x^{(i)}), y^{(i)}\Big) = \max\big(0, -y^{(i)} f(x^{(i)})\big) = \max\Big(0, -y^{(i)} \sum_{j=1}^{N_i} w_j x_j^{(i)}\Big) \tag{7.10}$$

Lorsque le réseau prédit le bon signe de la donnée ($signe(f(x^{(i)})) = signe(y^{(i)})$, donc $-y^{(i)} f(x^{(i)}) < 0$), la fonction d'erreur vaut 0, et il n'est pas nécessaire de corriger les poids puisque la prédiction est bonne. Lorsque le réseau se trompe, l'erreur est d'autant plus grande que le produit $|y^{(i)}|.|f(x^{(i)})|$ est grand : les poids doivent alors être modifiés pour réduire l'erreur en fonction des w_j. Or, dans ce cas, $\frac{\partial L}{\partial w_j} = -x_j^{(i)} y^{(i)}$, donc on peut corriger les poids, en utilisant un facteur de relaxation η, tel que :

$$\begin{cases} w_j = w_j & \text{si} \quad y^{(i)} f(x^{(i)}) > 0 \\ w_j = w_j + \eta x_j^{(i)} y^{(i)} & \text{si} \quad y^{(i)} f(x^{(i)}) < 0 \end{cases} \tag{7.11}$$

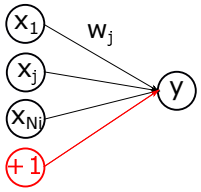


FIGURE 7.10 – Perceptron simple.

Cas de la régression

Pour la régression sur ce même perceptron simple, l'erreur utilisée est généralement l'écart quadratique :

$$L\Big(f(x^{(i)}), y^{(i)}\Big) = \frac{1}{2}\Big(y^{(i)} - f(x^{(i)})\Big)^2 = \frac{1}{2}\Big(y^{(i)} - \sum_{j=1}^{N_i} w_j x_j^{(i)}\Big)^2 \tag{7.12}$$

L'actualisation des poids s'écrit alors :

$$w_j = w_j + \eta\Big(y^{(i)} - f(x^{(i)})\Big) x_j^{(i)} \tag{7.13}$$

Une forme similaire d'actualisation peut être utilisée pour la classification multi-classe ou binaire lorsqu'un critère d'entropie croisée est considéré.

Extension aux perceptrons multicouches

De manière générale, la réactualisation des poids lors de l'entrainement vise à réduire la fonction d'erreur : le terme qui est ajouté à la valeur actuelle de w_j est donc lié au gradient $\frac{\partial L}{\partial w_j}$ mais avec un signe négatif pour se déplacer dans le sens décroissant de la fonction. On parle souvent de méthodes de descente de gradient. Dans leur principe général, les méthodes d'entrainement des réseaux de neurones s'inspirent des méthodes d'optimisation par descente de gradient, mais avec un frein important lié à un facteur de relaxation.

L'originalité de ces méthodes provient des propriétés de ce gradient dûes à la structure même du réseau. L'erreur L dépend indirectement de w_j : entre le poids w_j et la sortie $f(x^{(i)})$, il peut y avoir plusieurs couches, donc plusieurs sommes pondérées combinées aux fonctions d'activation des différentes couches de neurones. Si l'expression analytique de cette dérivée semble humainement inaccessible, il se trouve que la règle de la chaîne permet de grandement la simplifier.

Si on considère le réseau multicouche de la Figure 7.11, avec des activations sigmoïdes sur les couches cachées et une activation linéaire sur la couche de sortie, on peut écrire les relations suivantes liées à la somme pondérée dans chaque couche et à l'application de la fonction d'activation dans chaque couche :

$$f(x_i^{(d)}) = o^{(3)} = z_m^{(3)} = \sum_{k=1}^{N_{h,2}} w_{km}^{(3)} o_k^{(2)} \tag{7.14}$$

$$o_k^{(2)} = \sigma\big(z_k^{(2)}\big) = \sigma\Big(\sum_{j=1}^{N_{h,1}} w_{jk}^{(2)} o_j^{(1)}\Big) \tag{7.15}$$

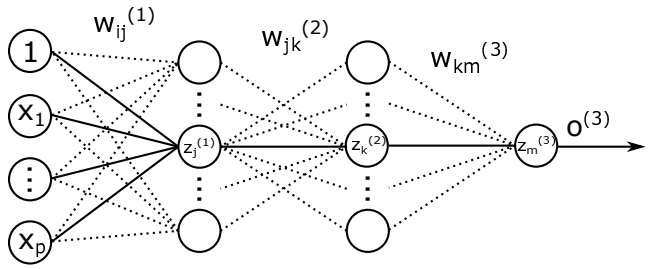


FIGURE 7.11 – Schéma d'un perceptron multicouche.

$$o_j^{(1)} = \sigma\big(z_j^{(1)}\big) = \sigma\left(\sum_{i=1}^{N_i} w_{ij}^{(1)} x_i\right) \tag{7.16}$$

Et la fonction d'erreur est :

$$L\Big(f(x_i^{(d)}); y^{(d)}\Big) = \frac{1}{2}\Big(f(x_i^{(d)}) - y^{(d)}\Big)^2 \tag{7.17}$$

Sur cette base, on peut calculer les gradients des poids de la dernière couche :

$$\frac{\partial L}{\partial w_{km}^{(3)}} = \frac{\partial\Big(\frac{1}{2}\big(f(x_i^{(d)}) - y^{(d)}\big)^2\Big)}{\partial w_{km}^{(3)}} = \big(f(x_i^{(d)}) - y^{(d)}\big)\frac{\partial f(x_i^{(d)})}{\partial w_{km}^{(3)}} = \big(f(x_i^{(d)}) - y^{(d)}\big)o_k^{(2)} \tag{7.18}$$

Il faut garder à l'esprit que la réactualisation des poids est effectuée après avoir présenté la donnée $\{x_i^{(d)}; y^{(d)}\}$ au réseau et avoir calculé la sortie $f(x_i^{(d)})$. Lors de ce calcul, toutes les valeurs intermédiaires des couches cachées ($z_j^{(1)}$, $o_j^{(1)}$, $z_k^{(2)}$ et $o_k^{(2)}$) ont été calculées : on dispose donc de toutes les valeurs pour calculer ce gradient. On peut donc alors directement réactualiser tous les poids $w_{km}^{(3)}$ tel que :

$$w_{km}^{(3)} = w_{km}^{(3)} - \eta\big(f(x_i^{(d)}) - y^{(d)}\big)o_k^{(2)} \tag{7.19}$$

On peut ensuite chercher à actualiser la seconde couche des poids $w_{jk}^{(2)}$:

$$\frac{\partial L}{\partial w_{jk}^{(2)}} = \frac{\partial L}{\partial f(x_i^{(d)})}.\frac{\partial f(x_i^{(d)})}{\partial o^{(3)}}.\frac{\partial o^{(3)}}{\partial o_k^{(2)}}.\frac{\partial o_k^{(2)}}{\partial z_k^{(2)}}.\frac{\partial z_k^{(2)}}{\partial w_{jk}^{(2)}} \text{ par la règle de la chaîne} \tag{7.20}$$

$$\frac{\partial L}{\partial w_{jk}^{(2)}} = \big(f(x_i^{(d)}) - y^{(d)}\big).1.w_{km}^{(3)}.\frac{\partial o_k^{(2)}}{\partial z_k^{(2)}}.o_j^{(1)} \tag{7.21}$$

La dérivée $\dfrac{\partial o_k^{(2)}}{\partial z_k^{(2)}}$ dépend de la fonction d'activation de la deuxième couche cachée. Or, la fonction sigmoïde possède une propriété très utile dans ce cas : $\sigma'(u) = u'\sigma(u)\big(1 - \sigma(u)\big)$, donc :

$$\frac{\partial o_k^{(2)}}{\partial z_k^{(2)}} = \frac{\partial \sigma(z_k^{(2)})}{\partial z_k^{(2)}} = \sigma'(z_k^{(2)}) = \sigma(z_k^{(2)})\big(1 - \sigma(z_k^{(2)})\big) = o_k^{(2)}\big(1 - o_k^{(2)}\big) \tag{7.22}$$

d'où :

$$\frac{\partial L}{\partial w_{jk}^{(2)}} = \big(f(x_i^{(d)}) - y^{(d)}\big).w_{km}^{(3)}.o_k^{(2)}\big(1 - o_k^{(2)}\big).o_j^{(1)} \tag{7.23}$$

Toutes ces grandeurs sont connues ou ont été calculées lors de la progration avant dans le réseau. On peut donc actualiser les poids $w_{jk}^{(2)}$ tels que :

$$w_{jk}^{(2)} = w_{jk}^{(2)} - \eta\big(f(x_i^{(d)}) - y^{(d)}\big).w_{km}^{(3)}.o_k^{(2)}\big(1 - o_k^{(2)}\big).o_j^{(1)} \tag{7.24}$$

Cette même logique peut être prolongée pour remonter progressivement vers les couches précédentes, mais les expressions de dérivées augmentent en complexité. Cette logique de réactualisation à contre-courant est le principe de base de la rétropropagation.

7.4.4 Vue générale et comparaison des méthodes d'entrainement

Le Tableau 7.3 présente une liste de méthodes utilisées pour l'entrainement de réseaux de neurones, ainsi que leurs noms en termes d'implémentation dans Matlab. On y retrouve la rétropropagation, ainsi que de nombreuses méthodes inspirées de l'optimisation mais spécifiquement adaptées à la forme particulière des réseaux de neurones (gradient conjugué, Quasi-Newton, etc.). Les différentes méthodes se distinguent par leur besoin de stockage en mémoire et leur vitesse de convergence.

Méthode	Implémentation Matlab	Stockage	Vitesse
Rétropropagation	`train`		
Descente de gradient	`traingd`		
Descente de gradient avec inertie	`traingdm`		
Algorithme à taux d'apprentissage adaptatif	`traingda, traingdx`	+	- -
Rétropropagation résiliente	`trainrp`	+	+ + +
Gradient conjugué (Fletcher-Reeves)	`traincgf`	+	
Gradient conjugué (Polak-Ribière)	`traincgp`	-	+
Gradient conjugué (Powell-Beale)	`traincgb`	- -	+ +
Gradient conjugué ("scaled")	`trainscg`		+ +
Méthode de Quasi-Newton (BFGS)	`trainbgf`	-	+ +
Méthode de Quasi-Newton (One Step Secant)	`trainoss`		
Levenbergh-Marquardt	`trainlm`	- - -	+ + +

TABLE 7.3 – Principales méthodes numériques utilisées pour la rétropropagation et l'optimisation des poids et biais d'un réseau. Les symboles + et − indiquent la qualité des méthodes selon les critères de stockage en mémoire et de vitesse de convergence.

La méthode de Levenberg-Marquardt est très souvent citée comme étant la plus efficace (Khalil Arya et Ayati, 2013). Néanmoins, la méthode optimale dépend fortement du problème à résoudre. Pour les problèmes d'approximation de fonction, avec des réseaux contenant jusqu'à quelques centaines de poids, l'algorithme de Levenberg-Marquardt présente la convergence la plus rapide et atteint des erreurs quadratiques moyennes inférieures aux autres algorithmes. Lorsque le nombre de poids augmente, cet algorithme perd en efficacité et ses performances deviennent médiocres sur les problèmes de reconnaissance de formes.

La méthode de rétropropagation résiliente est la plus rapide sur les problèmes de reconnaissance de formes, mais elle n'est pas efficace pour l'approximation de fonction. Ses besoins de stockage sont faibles par rapport aux autres algorithmes considérés.

Les méthodes de gradient conjugué fonctionnent bien sur une grande variété de problèmes, en particulier pour les réseaux avec un grand nombre de poids. L'algorithme SCG est presque aussi rapide que Levenberg-Marquardt sur les problèmes d'approximation, est plus rapide pour les grands réseaux et presque aussi rapide que la rétropropagation résiliente sur les problèmes de reconnaissance de formes.

Les performances de la méthode BFGS sont similaires à celles de Levenberg-Marquardt. Elle nécessite moins de stockage, mais le temps de calcul augmente géométriquement avec la taille du réseau, car l'équivalent d'une matrice inverse doit être calculé à chaque itération.

L'algorithme à taux d'apprentissage adaptatif est généralement beaucoup plus lent que les autres méthodes et a environ les mêmes exigences de stockage que la rétropropagation résiliente. Mais il peut toujours être utile pour certains problèmes, où il vaut mieux converger plus lentement pour ne pas dépasser le point où l'erreur de validation est minimisée.

Elles peuvent toutes être appliquées selon deux modes différents :

— En **mode incrémental**, le gradient est calculé et les poids sont mis à jour après l'application de chaque donnée au réseau.

— En **mode batch**, toutes les entrées sont appliquées au réseau avant la mise à jour des poids.

7.5 Post-traitement, exploitation et réduction de réseaux de neurones

Après entrainement et validation d'un réseau, il peut être exploité pour traiter de nouvelles données. Cette phase d'exploitation peut nécessiter un post-traitement pour comprendre le modèle qu'il décrit ou réduire la taille du réseau, afin de faciliter son utilisation.

7.5.1 Analyse de sensibilité par rapport aux entrées

Alors que les méthodes statistiques comme la régression multilinéaire permettent de quantifier des intervalles de confiance sur les paramètres du modèle pour savoir quelles variables ont le plus d'impact sur la sortie, les réseaux de neurones ne permettent pas d'obtenir ce type d'information. On doit donc calculer des indices pour se faire une idée de ces influences relatives.

Leite et al. (2019) utilisent un critère d'importance relative $RI_{i,k}$ de l'entrée i par rapport à la sortie k (pour des perceptrons monocouches) défini tel que, si on utilise l'indice i pour les N_{input} neurones de la couche d'entrées, j pour les N_{hidden} neurones de la couche cachée et k pour les N_{output} neurones de la couche de sortie :

$$RI_{i,k} = \frac{\left|\sum_{j=1}^{N_{hidden}} w_{i,j} w_{j,k}\right|}{\left|\sum_{i=1}^{N_{input}} \sum_{j=1}^{N_{hidden}} w_{i,j} w_{j,k}\right|} \tag{7.25}$$

7.5.2 Problème de l'explicabilité et de l'interprétabilité d'un réseau

La question de l'explicabilité n'est pas spécifique aux réseaux de neurones, mais concerne la plupart des méthodes algorithmiques d'intelligence artificielle. Dans un contexte beaucoup plus large que le Génie des Procédés, l'utilisation d'algorithmes (notamment d'intelligence articielle) soulève de nombreuses questions d'éthique : la législation française (Loi pour une république numérique) impose de *devoir expliquer une décision administrative obtenue par un traitement automatique*. Cela peut concerner l'emploi, l'éducation, la santé, l'assurance, le traitement judiciaire, etc.

On distingue l'explicabilité et l'interprétabilité :

— Une décision algorithmique est dite *explicable* s'il est possible d'en rendre compte explicitement à partir des données et caractéristiques connues de la situation, c'est-à-dire s'il est possible de relier les valeurs de certaines variables et leurs conséquences sur la prévision, et ainsi sur la décision. On retrouve l'idée "de cause à effet".

— Une décision algorithmique est dite *interprétable* s'il est possible d'identifier les variables qui participent le plus à la décision, voire même d'en quantifier l'importance. On retrouve l'idée de sensibilité.

En Génie des Procédés, le problème de l'explicabilité se traduit, dans un premier temps, par la recherche des variables influentes grâce à une étude de sensibilité. Dans un second temps, c'est la relation entre ces variables influentes et les sorties prédites qui est étudiée. Pour beaucoup de méthodes, puisque la sortie est calculable à partir des entrées, il existe une procédure, voire une expression, de calcul, éventuellement très complexe. La difficulté consiste alors à comprendre cette relation, voire à tenter de la relier à une chaîne de causes et d'effets physiques, au moins dans ses grandes tendances.

Cette étape d'interprétation est quasiment vaine lorsque certaines étapes de la méthode incluent des méthodes stochastiques (forêts aléatoires), de fortes sensibilités aux conditions initiales (K-moyennes), des optima multiples (réseaux de neurones) ou des réductions de dimensionnalité par projection sur des variables latentes non-linéaires (réseaux auto-encodeurs, cartes auto-adaptatives).

L'explicabilité et ses conséquences restent aujourd'hui une question ouverte.

7.5.3 Réduction de réseaux

Lorsqu'un réseau de neurones a été entrainé, il peut être utile de le simplifier : en effet, il est possible que certains neurones aient un effet négligeable sur les sorties. L'étude des poids synaptiques renseigne sur cette influence : si un poids $w_{j,k}$ entre la couche $N_{h,j}$ et la couche $N_{h,k}$ est nul, alors la sortie du neurone $N_{h,j}$ sera multipliée par 0 pour calculer la sortie du neurone $N_{h,k}$. Il n'est donc pas utile de conserver le neurone $N_{h,j}$. L'analyse des valeurs des poids renseigne donc sur la possibilité de réduire la taille du réseau.

D'autres algorithmes, appelés Pruning, Dropout, DropConnect, permettent de tester automatiquement ce manque d'influence pendant la phase d'entrainement. Le lecteur intéressé pourra consulter Goodfellow (2018) et Tufféry (2019).

7.6 Conseils et heuristiques

7.6.1 Choix des réseaux de neurones pour la régression

Le théorème d'approximation universelle qui caractérise les réseaux de neurones est une force incontestable, mais ne doit pas justifier un recours systématique à cette méthode. Si le recours à un réseau de neurones se

comprend pour des situations complexes, l'utilisation d'outils classiques tels que la régression multilinéaire (voir section 2.2) doit être préférée pour des problèmes simples :

- Pour des modèles simples (polynomiaux) en faible dimension, la régression multilinéaire requiert moins de paramètres inconnus qu'un réseau de neurones : un polynôme de degré n (d'une seule variable) requiert $n+1$ paramètres alors qu'un réseau de neurones équivalent pourra nécessiter plusieurs dizaines de poids synaptiques. Le nombre de données nécessaires est donc plus faible.
- la régression multilinéaire fournit des valeurs déterministes et optimales des paramètres au sens des moindres carrés, il n'y a pas de multiplicité des solutions optimales.
- la régression multilinéaire ne nécessite pas d'entrainement.
- la régression multilinéaire n'est pas sujette au sur-apprentissage.
- la régression multilinéaire permet de calculer des intervalles de confiance sur les paramètres pour simplifier le modèle.
- la régression multilinéaire peut être appliquée avec des données en flux : la régression récursive est présentée dans le chapitre 9.
- la régression multilinéaire n'est pas plus difficile à construire ou à exploiter qu'un réseau de neurones : elle ne résume à une equation généralement simple.
- la régression permet une explicabilité plus simple du résultat obtenu.
- la régression multilinéaire ne nécessite pas d'optimiser des hyperparamètres ni de choisir une méthode d'entrainement. Par contre, elle nécessite de réfléchir à l'expression mathématique du modèle qu'on souhaite régresser.

La régression multilinéaire doit donc être testée avant de mettre en oeuvre une méthode basée sur des réseaux, car, si une régression simple est accessible, elle fournira la réponse beaucoup plus vite et permettra une meilleure compréhension de la régression obtenue.

Trois principaux cas de figures justifient l'accès à une régression par réseau de neurones :

- Lorsque les expressions simples de modèles échouent à régresser les données : Si plusieurs expressions de modèles ont été tentées mais qu'aucune ne permet de corréler proprement les données, alors il peut être très chronophage de chercher une expression satisfaisante. Il est néanmoins possible de construire des régressions "aveugles" où chaque terme du modèle prend la forme générique $x_1^{\alpha_1} x_2^{\alpha_2} x_3^{\alpha_3} \dots x_p^{\alpha_p}$ en balayant des plages pour α_1, α_2, $\dots$ α_p et en ne retenant finalement que les termes dont les paramètres sont significatifs. Ces modèles deviennent vite problématiques en raison de la taille des matrices Φ et $\Phi^T.\Phi$ qui doit notamment être inversée.
- Lorsque des non-linéarités fortes sont attendues entre les variables et la sortie prédite : la régression multilinéaire peut régresser des non-linéarités par rapport aux variables comme des termes en ax_1^2, ax_1x_2, $a\frac{x_3}{\log(1-x_7)}$ avec a inconnu, mais ne peut pas régresser des termes en a^{x_4} ou $\cos(x_1+ax_2)$ avec a inconnu, alors que les réseaux de neurones peuvent réussir à construire une somme pondérée qui approximera ces fonctions hautement non-linéaires.
- Lorsque la dimensionnalité du problème est grande : lorsque la dimension p de l'espace des variables devient grande, alors la régression multilinéaire commence à perdre sa facilité, car le nombre de termes à inclure dans un modèle simple explose. En 3 dimensions, un modèle basique est $y = a_0+a_1x_1+a_2x_2+a_3x_3$. On peut le complexifier avec 3 termes x_i^2 $(1 \leq i \leq 3)$ ou 3 termes x_ix_j, et éventuellement $x_1x_2x_3$, on a alors 11 termes. Le nombre de termes nécessaires augmente exponentiellement avec la dimension p et la régression multilinéaire perd en compétitivité par rapport aux réseaux de neurones.

7.6.2 Choix des hyper-paramètres du réseau

Au moins deux hyperparamètres doivent être choisis ou testés pour mettre en oeuvre un réseau de neurones : le nombre de couches cachées et le nombre de neurones dans ces couches. Les fonctions d'activation peuvent aussi faire l'objet d'un réglage.

Concernant le nombre de couches cachées, une seule couche intermédiaire est souvent suffisante pour la régression et la classification car cette couche unique permet de représenter n'importe quelle fonction (théorème d'approximation universelle). L'ajout de couches cachées intermédiaire n'a d'utilité que lorsque les données de départ possèdent une structure inconnue, mais il faut alors fonctionnaliser ces couches et cela relève du Deep Learning.

Concernant le nombre de neurones à placer dans la couche cachée, il est généralement dépendant du nombre de données :

- si les données d'entrée ne sont pas bruitées (cas de l'approximation de fonctions analytiques), alors il faut 2 à 5 fois plus de données que de neurones dans le réseau (Ihme et al., 2006)).
- si les données sont bruitées, il faut plutôt 10 à 30 fois plus de données que de neurones (Khalil Arya et Ayati, 2013)

Ces ordres de grandeur permettent d'éviter deux biais de la méthode. S'il n'y a pas assez de neurones, le réseau est en sous-apprentissage, il ne possède pas une structure lui permettant de représenter la complexité des données. S'il y a trop de neurones, donc trop de poids synaptiques à entrainer, le réseau possède beaucoup de degrés de liberté et il peut représenter n'importe quoi : il parvient certes à représenter les grandes tendances des données, mais il représente aussi les incertitudes et les erreurs associées. Le réseau est en sur-apprentissage et il ne sera pas fiable sur des données nouvelles qu'il n'a jamais vues.

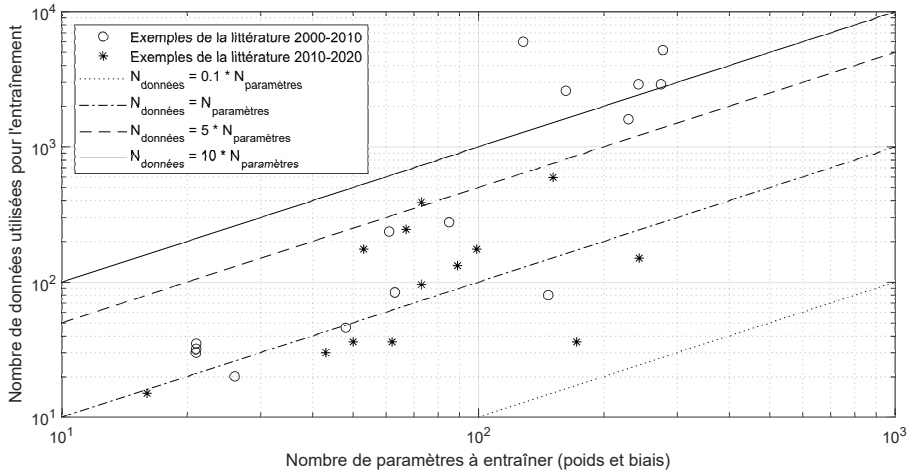


FIGURE 7.12 – Comparaison de travaux récents de la littérature avec les heuristiques reliant le nombre de données d'entraînement et le nombre de paramètres à entraîner.

7.6.3 Autres conseils

Il est conseillé de normaliser les données d'entrées x_i pour qu'elles aient des valeurs entre -1 et 1. Cela réduit les problèmes d'arrondis numériques qui induiraient de mauvaises estimations, et cela permet surtout, à la vue des poids et biais, de se faire une idée rapide des entrées qui ont des effets négatifs ou positifs, et des effets majeurs ou négligeables. Cela peut permettre de simplifier le réseau par les techniques de "Dropout" mentionnées précédemment, mais cela ne remplace pas une analyse de sensibilité entre les entrées et les sorties du réseau.

Il ne faut jamais utiliser un réseau entrainé sans lui avoir fait passer la phase de test (voir section 3.4). Le lot de données initiales doit toujours être fractionné en au moins deux parties, les données d'entrainement et les données de test, pour estimer l'erreur de généralisation du réseau. L'entrainement et le test doivent être généralement menés de front pour éviter le surapprentissage.

7.7 Applications en Génie des Procédés

Beaucoup d'applications en Génie des Procédés visent à développer des modèles représentatifs de données sans nécessiter la construction ou l'écriture de modèles mathématiques : leur utilisation se résume souvent à du "lissage" (fitting) efficace mais non explicatif.

Une fois entraîné et validé, un réseau de neurones peut très rapidement calculer ses sorties en fonction de nouvelles entrées : c'est donc un outil particulièrement intéressant pour des applications en temps réel, notamment en contrôle de procédés. Cette caractéristique est aussi utile pour de l'optimisation de procédés par des logiciels de CPAO, où un modèle détaillé et chronophage d'une opération unitaire peut être remplacé par un modèle de substitution (réseau de neurones ou autre) moins précis mais beaucoup plus rapide à "résoudre".

7.7.1 Chimie, synthèse, et catalyse

Gasteiger et Zupen (1993) montrent l'étendue des développements des réseaux de neurones pour des applications en chimie, via des études de régression par perceptrons monocouches ou des classifications par cartes auto-adaptatives.

Rothenberg (2008) passe en revue les différentes méthodes utilisées en catalyse, notamment les réseaux de neurones.

Maltarollo et al. (2013) présentent une revue générale des application des réseaux de neurones sur divers problèmes de chimie (chimie médicale et pharmaceutique, chimie théorique, chimie analytique, biochimie, etc.).

Li et al. (2017) passent en revue plusieurs applications de réseaux de neurones en catalyse pour la modélisation et l'optimisation des catalyseurs et des réactions catalytiques.

Fernandez et al. (2017) utilisent des réseaux de neurones pour régresser l'activité et la sélectivité catalytique de nanoparticules de platine sur la base de descripteurs identifiés par réduction de dimensionnalité. Les réseaux testés vont de 2-20-3 à 2-40-3, les deux entrées étant les deux descripteurs de structure et les 3 sorties étant l'activité, la sélectivité et la stabilité du catalyseur.

7.7.2 Chimie analytique

Zivkovic et al. (2009) entrainent un perceptron monocouche 6-6-1 (48 paramètres) avec 46 données d'entrainement et 21 données de test pour déterminer la teneur en cuivre de déchets et comparent leur résultat à une régression multilinéaire sans effets croisés ou quadratiques.

7.7.3 Matériaux

Gunesoglu et Kaplangiray (2019) entrainent un perceptron 5-6-2 (50 paramètres) avec 36 données pour corréler les propriétés thermiques de tissus tricotés en fonction de leur poids, leur épaisseur, la densité de fibres et leur conductivité.

Oguntade et al. (2020) conçoivent un perceptron 3-3-1 (16 paramètres), entrainé sur 15 expériences (mais sans phase de validation) pour corréler la vitesse de corrosion d'un acier en fonction du pH, du temps et de la concentration d'un inhibiteur constitué d'huile de castor et de sésame. Ils comparent leur résultat à une régression multi-linéaire à 10 paramètres issue d'un plan factoriel de Box-Behnken à 3 facteurs sur les 15 mêmes expériences.

7.7.4 Propriétés physiques de composés et de mélanges

Otero (1998) entraîne un perceptron 3-10-1 pour prédire la diffusivité moléculaire de surfactants dans des systèmes micellaires.

Iliuta et al. (2000) entrainent deux perceptrons 12-18-2 (274 paramètres) et 12-16-2 (242 paramètres) avec 2900 données pour prédire les grandeurs d'équilibre liquide-vapeur de mélanges solvant-électrolyte : ces réseaux contiennent au final beaucoup moins de paramètres que les modèles UNIQUAC équivalents.

Chouai et al. (2002) entraînent un perceptron monocouche pour prédire le facteur de compressibilité Z de 3 fluides réfrigérants en fonction des T et P. Ils exploitent aussi le réseau pour calculer les dérivées $\frac{dZ}{dT}$ pour accéder à l'enthalpie et l'entropie de la phase liquide. Les données d'entraînement sont issues de REFPROP.

Ganguly (2003) développent deux réseaux de neurones pour prédire les propriétés d'équilibres liquide-vapeur de 4 mélanges binaires et deux ternaires.

Boozarjomehry et al. (2005) testent 5 perceptrons de 2-3-1 à 2-8-1 (13 à 33 paramètres) pour corréler la température critique, la pression critique, le volume critique, le facteur acentrique et la poids moléculaire de composés purs et de fractions pétrolières en fonction de leur point d'ébullition et leur densité liquide. Malgré l'optimisation, ils observent que les réseaux de neurones ne sont pas toujours plus précis que des modèles plus conventionnels.

Lisa et Curteanu (2005) entrainent un perceptron 2-5-1 (21 paramètres) avec 35 données d'entrainement et 5 de test pour corréler la masse volumique de mélanges binaires en fonction de la température et de la fraction molaire. Le réseau fournit la même qualité de prédiction des modèles multilinéaires ayant, au plus, 7 paramètres.

Anitha et Singh (2008) entraînent un réseau 3-5-1 (26 paramètres) avec 20 données pour corréler les coefficients de partition de néodyme et praséodyme en vue de leur extraction par solvant.

Eslamloueyan et al. (2009) construisent des réseaux de neurones pour estimer la conductivité thermique de gaz : un réseau 4-10-1 (61 paramètres) avec 236 données pour des gaz purs et un réseau 5-12-1 (85 paramètres) avec 277 données pour des mélanges binaires.

Kurt et Kayfeci (2009) entraînent un perceptron 3-4-1 (21 paramètres) sur 32 données pour prédire la conductivité thermique de mélanges eau-éthylène glycol.

Moghadassi et al. (2010) entraînent un réseau à 1 couche cachée de 19 neurones pour prédire la viscosité de différents liquides.

Eslamloueyan et al. (2011) construisent un perceptron 4-12-1 (73 paramètres) avec 389 données pour corréler la conductivité thermique de solutions d'électrolytes aqueux. Le réseau obtenu présente de bonnes caractéristiques d'extrapolabilité et d'intrapolabilité à d'autres électrolytes que ceux considérés pendant l'entraînement.

Sabzevari et Moosavi (2014) utilisent un perceptron monocouche 8-15-1 (151 paramètres) et 595 données pour corréler la masse volumique de métaux alcalins liquides en fonction de leur composition, la température et la pression.

Alabi et Williamson (2015) entraînent un réseau 3-7-1 (36 paramètres) pour corréler la viscosité de la "liqueur noire" (liqueur de cuisson issue de la fabrication du papier kraft).

Li et Zhang (2018) développent une perceptron monocouche pour prédire la solubilité du CO_2 dans des mélanges.

7.7.5 Modélisation et optimisation de réacteurs et de procédés

Patnaik (1999) présente une variété de cas d'application de réseaux de neurones pour des méthodes de séparation de produits biologiques.

Chouai et al. (2000) entraînent un perceptron monocouche 11-9-2 (128 paramètres) avec 6000 données pour modéliser un procédé d'extraction liquide-liquide.

Nascimento et al. (2000) développent des réseaux de neurones soit pour créer des modèles non-linéaires à partir de données, soit pour créer des modèles de substitution et remplacer des modèles phénoménologiques trop chronophages dans des optimisations de procédés. Ils entraînent notamment un réseau 13-10-2 (162 paramètres) avec 2600 données pour un procédé de production d'anhydride acétique.

Razavi et al. (2003) entraînent un réseau 2-10-3 (63 paramètres) avec 84 données pour modéliser une étape d'ultrafiltration de lait.

Al-Hemiri et Salih (2007) entraînent un perceptron monocouche 3-4-1 (21 paramètres) avec une trentaine de données pour corréler les coefficients de transfert de matière $k_L.a$ dans une colonne à bulles.

Khataee et al. (2011) entraînent un perceptron 5-14-1 (89 paramètres) avec 132 données pour corréler un taux d'extraction de colorant de microalgues en fonction du pH, de la température, du temps, etc.

Pandharipande et al. (2012) entraînent et comparent deux perceptrons monocouches 3-5-5-2 (62 paramètres) et 3-10-10-2 (172 paramètres) grâce à 36 expériences d'entraînement et 12 expériences de tests pour modéliser le transfert de matière dans des colonnes à spray pour une opération d'extraction liquide-liquide.

Khalil Arya et Ayati (2013) entraînent un perceptron multi-couche 3-6-6-1 (73 paramètres) sur 96 données pour prédire l'efficacité d'un traitement d'hydroquinone dans des contacteurs à disques ou lits tournants. Ils comparent 13 méthodes d'entraînement du réseau et retiennent la méthode de Levenbergh-Marquardt.

Santos et al. (2013) entraînent un perceptron monocouche 9-20-1 à 221 paramètres pour construire un modèle en vue de la régulation d'un procédé batch de polymérisation de styrène.

Basile et al. (2015) utilisent un perceptron monocouche pour modéliser et optimiser un réacteur de water gas shift. Le réseau est constitué de 8 entrées, 20 neurones dans la couche cachée et 3 sorties. 150 mesures expérimentales sont utilisées pour entraîner le réseau (243 poids et biais), dont 60 % pour l'entraînement, 20 % pour le test et 20 % pour la validation.

Prakash Maran et Priya (20105) entraînent un réseau 4-7-1 (43 paramètres) avec 30 données pour modéliser un système de production de biodiesel à partir d'huile végétale. La comparaison avec une régression multilinéaire à seulement 15 paramètres fait logiquement pencher la balance dans le sens du réseau de neurones, en termes de précision de prédiction, mais il aurait été impossible de faire une telle régression avec 43 paramètres.

Hachemaoui et Balhamel (2017) entraînent un perceptron à 9 entrées, 6 neurones cachés et une sortie (67 paramètres), selon différentes différentes méthodes sur la base de 245 mesures pour régresser la concentration de nickel en sortie d'un procédé d'extraction en émulsion en fonction des conditions opératoires. Malgré une erreur relative moyenne de 12 %, minimisée en ajustant le nombre de neurones, certaines données présentent une erreur relative supérieure à 60 %.

Rosli et Aziz (2017) présentent une review des utilisations de réseaux de neurones pour modéliser le procédé de craquage : perceptrons multicouches, réseaux adaptatifs, réseaux récurrents, méthodes hybrides, etc.

Voyant et al. (2017) décrit les méthodes de régression utilisées pour la prédiction du rayonnement solaire dans les procédés impliquant du solaire photovoltaique ou thermique : régression linéaire et généralisée, réseaux de neurones, etc.

Leite et al. (2019) entraînent 2 perceptrons monocouches 5-8-1 (53 paramètres) et 5-14-1 (99 paramètres) avec 175 données pour modéliser un procédé d'adsorption pour la séparation du lactose dans du lait. Ils utilisent la fonction d'activation radiale.

7.7.6 Détection et diagnostic de fautes et dérives

Himmelblau et al. (1991) entraînent 2 perceptrons 5-4-2 et 4-10-1 pour diagnostiquer le fonctionnement d'échangeurs de chaleur et de machines tournantes.

Himmelblau et al. (2000) rapportent un cas de détection de bouchage dans un échangeur de chaleur à partir d'un réseau 8-12-3 (147 paramètres) entraîné avec 80 données.

Alves et Nascimento (2002) entraînent un perceptron multicouche pour détecter des points de fonctionnement aberrants parmi des données historiques d'un procédé de production d'isoprène.

Azhar et Rahman (2010) entraînent un perceptron mono-couche à 21 neurones cachés avec la méthode Levenbergh-Marquardt pour détecter des dysfonctionnements d'un procédé batch d'estérification.

7.7.7 Planification de production

Zhang et al. (2019) entraînent un réseau de neurones mono-couche pour prédire la popularité d'un nouveau produit, planifier des campagnes de production en atelier batch.

7.7.8 Contrôle, commande et régulation de procédés

Otero (1998) entraîne un perceptron 1-15-1 pour construire un modèle de régulation d'une cuve de mélange. La commande par modèle interne, basée sur le perceptron entraîné, s'avère finalement plus performante qu'un simple PID (apparemment mal réglé...) en suivi de consigne et en rejet de perturbations.

Mujtaba et al. (2006) contruisent une commande par modèle interne de procédés batch en régressant par réseaux de neurones le modèle et le modèle inverse du procédé.

Atasoy et al. (2007) testent une commande prédictive basée sur réseau de neurones sur un procédé de polymérisation d'acrylonitrile.

Fernandez de Canete et al. (2008) entrainent deux perceptrons multicouches 10-8-17-2 (277 paramètres) et 10-7-15-2 (229 paramètres) pour modéliser et réguler un procédé en mode multivariable, grâce à 5200 et 1600 données respectivement en boucle ouverte et boucle fermée.

Biyanto et al. (2010) développent une commande par modèle interne où le modèle de procédé est construit par réseau de neurones ainsi que le modèle inverse, et l'appliquent à une colonne à distiller.

Vasickaninova et al. (2020) construisent une commande prédictive basée sur réseaux de neurones et l'appliquent à une cascade de 4 échangeurs à contre-courant en série d'un procédé de refroidissement.

7.7.9 Développements méthodologiques

Ihme et al. (2006) applique une méthodologie pour l'optimisation des architectures de réseaux à un cas de combustion de méthane et cite de nombreuses références sur les méthodes évolutionnaires.

Issanchou et Gauchi (2008) proposent une méthode de planification expérimentale optimale pour améliorer la généralisation d'un réseau de neurones en minimisant le nombre d'expériences à réaliser.

Chapitre 8

Réseaux complexes et Deep Learning

Le "Deep Learning", ou apprentissage profond, est une catégorie de méthodes d'apprentissage automatique qui a explosé à partir des années 2010 grâce à la convergence de plusieurs facteurs : (i) les méthodes d'entrainement des réseaux de neurones multicouches permettaient enfin d'entrainer des réseaux en évitant l'évanouissement ou l'explosion du gradient, (ii) des ordinateurs suffisamment rapides étaient facilement accessible pour gérer en parallèle de grandes quantités de données à un coût abordable et (iii) de grandes bases de données étaient disponibles grâce à l'essor d'internet et à la mobilisation d'une large communauté d'internautes.

Le retour d'expériences sur l'utilisation des réseaux de neurones avait également permis une plus grande diversification des opérations effectuées par chaque couche cachée : la non-linéarité introduite par chaque couche n'était plus seulement attribuée aux fonctions d'activation. L'utilisation des réseaux de neurones en traitement d'images et en régression de données temporelles avait permis d'introduire les notions de convolutions et de mémoire dans les réseaux.

En quelques années, l'apprentissage profond a pris une place très importante dans beaucoup d'applications de la vie de tous les jours sans qu'on s'en rende compte : recommandations de produits, identification de profils utilisateurs, jeux en lignes, voiture autonome, reconnaissance faciale, traduction automatique, robots conversationnels, robotique avancée, diagnostic médical, etc. Ce domaine est en pleine explosion et beaucoup de ses applications potentielles sont encore inconnues car ces techniques n'ont pas eu le temps de diffuser dans beaucoup de domaines et d'être adaptées à des formalismes ou des contraintes spécifiques.

L'objectif du présent chapitre n'est donc pas de donnéer une vision détaillée, ni exhaustive, de ces techniques, mais de présenter succinctement les idées fortes qui se cachent derrière les méthodes les plus courantes utilisées en Génie des Procédés.

La principale caractéristique des réseaux profonds est leur fonctionnement en couches fonctionnalisées multiples : cet enchainement de couches permet d'identifier, d'extraire et de régresser des structures au sein des données d'entrée. Les fonctions élémentaires que savent assurer des réseaux monocouches (clustering, classification, régression) sont ainsi enchaînées sur des structures à différentes échelles dans les données originales. Un réseau de neurones appliqué à une photo pourra ainsi extraire et identifier les éléments qui se trouvent sur cette photo établir des associations entre ces objets, comparer ces associations à des références issues d'un entrainement préalable et déduire le contexte dans lequel cette photo a été prise, tout en reconnaissant nominativement les humains qui y apparaissent.

8.1 Réseaux convolutionnels

Les réseaux convolutionnels sont très utilisés en traitement d'image car ils permettent d'extraire des caractéristiques locales de la structure de l'image (contraste, frontière, gradient, ligne, etc.). La Figure 8.1 présente un exemple emblématique de leur utilisation pour l'identification automatique de chiffres manuscrits, développé initialement pour le tri automatique du courrier par la lecture des codes postaux figurant sur les enveloppes.

Ce réseau à 7 opérations reçoit une image de taille prédéfinie (28×28) et renvoit un entier entre 0 et 9 correspondant à la réponse du déchiffrement. Il inclut 2 opérations de convolution, 2 couches de pooling, une couche d'applatissement (flatten) et 2 couches denses neuronales classiques qui constituent une classification multi-classe. A chaque étape, la taille et la forme des données changent :

— chaque opération de convolution fait augmenter la taille des données en appliquant plusieurs filtres aux données précédentes : la première couche de la Figure 8.1 applique 32 filtres à l'image originale et génère donc 32 images de taille équivalente.

— chaque étape de pooling réduit la taille des données en ne retenant que la partie la plus significative d'une partie des données précédentes : dans l'exemple, la taille est divisée par 4 à chaque application car

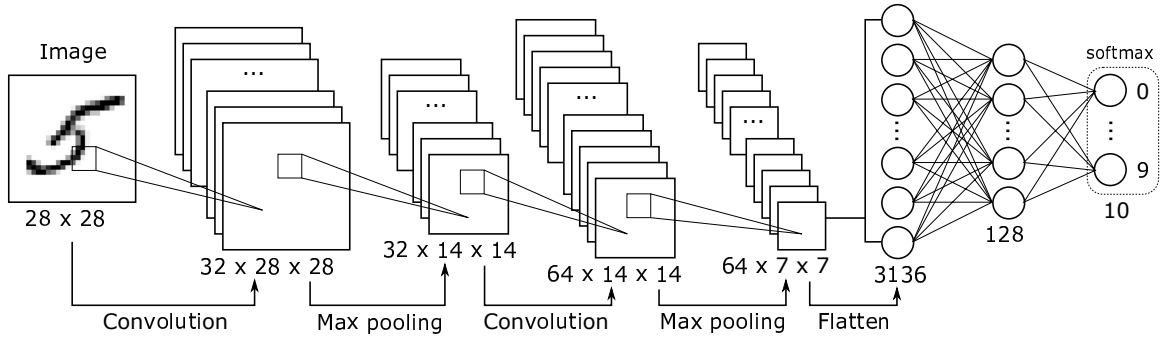


FIGURE 8.1 – Architecture d'un réseau convolutionnel pour l'identification de chiffres manuscrits.

le max pooling ne retient ici qu'un pixel sur 4 dans les images précédentes.

— l'étape de flattening ne modifie pas les données, elle les réorganise juste : les 3136 pixels qui constituent les 64 images (7 pixels × 7 pixels) en un vecteur de 3136 réels qui sert d'entrée au classificateur neuronal.

Les opérations de convolution et de pooling sont présentées ci-dessous.

8.1.1 Convolution

Contrairement aux couches des réseaux de neurones classiques qui sont constituées d'un très grand nombre de poids inconnus, les connexions entre "neurones" d'une couche de convolution possèdent beaucoup moins de diversité .

D'une part, cette couche de convolution n'est pas une couche dense : tous les neurones de chaque couche ne sont pas reliés entre eux (Figure 8.2, gauche). Chaque neurone de la couche de sortie n'est relié qu'à un certain nombre de neurones de la couche d'entrée, et ce nombre, ainsi que leurs positions relatives, sont les mêmes pour tous les neurones de sortie. De plus, les poids reliant un neurone de sortie à ses antécédents de la couche précédente sont les mêmes pour tous les neurones : la transformation est invariante par translation.

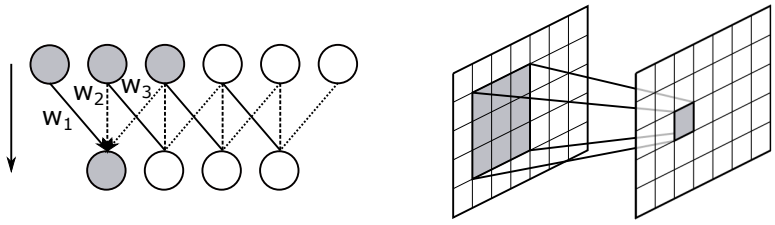


FIGURE 8.2 – Principe d'une opération de convolution entre deux couches de neurones.

Dans le cas de l'analyse d'image (Figure 8.2, droite), chaque couche de neurones est en fait une image, donc une matrice (avec éventuellement des sous-couches RGB pour les couleurs). Pour chaque pixel de l'image de sortie, la convolution s'effectue donc à partir d'un voisinage de pixels/neurones de l'image précédente. Le calcul de la valeur du pixel/neurone de sortie se fait selon une pondération des pixels/neurones antédécents par une matrice, semblable à celles présentées sur la Figure 8.3.

Selon les coefficients de cette matrice de convolution, différents effets sont appliqués à l'image : floutage uniforme ou sélectif, repoussage dans une direction préférentielle, détection de bords (contrastes, couleurs), renforcement de contraste, etc. Ces matrices (ici de taille 3×3) peuvent être de taille supérieure. Il existe de nombreuses variantes de ces matrices de convolution (Goodfellow, 2018 ; Tufféry, 2019).

Dans le cas de la Figure 8.1, la première étape de convolution applique 32 matrices différentes à l'image originale pour générer 32 nouvelles images qui représentent chacune une informations paritculière de l'image originale.

8.1.2 Max Pooling

Le pooling est une opération simple qui consiste à rempalcer un groupe de pixels/neurones (2×2) par une valeur unique, extraite ou calculée à partir des valeurs originales (Figure 8.4). Cette étape permet de réduire significativement la taille des images sans perdre les caractéristiques les plus fortes. Le choix des caractéristiques à conserver requiert de choisir parmi plus possibilités, parmi lesquelles on peu citer :

— le Max pooling : on ne retient que la plus grande valeur parmi les pixels originaux.

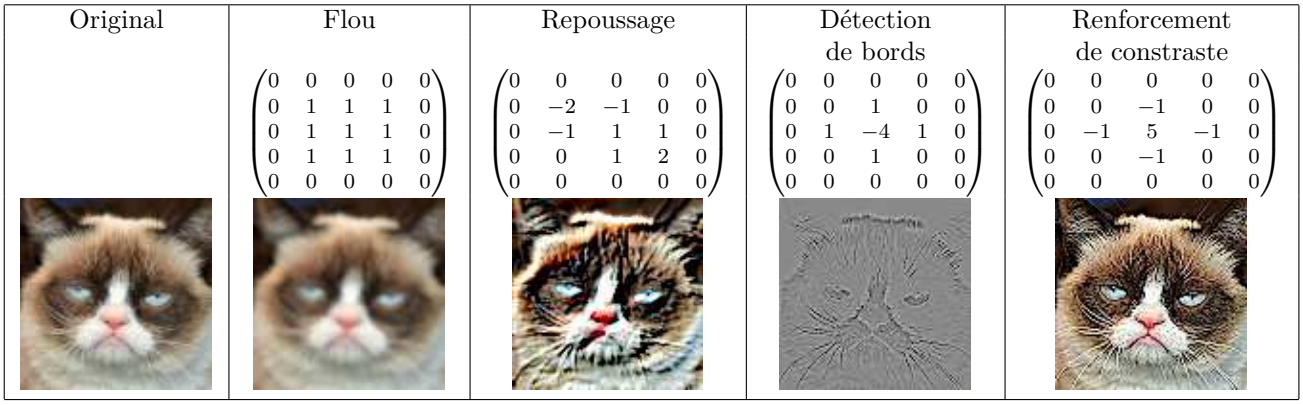


FIGURE 8.3 – Exemples de résultats de convolutions 3×3 appliquées à une image.

— le Mean pooling : on calcule la moyenne des valeurs.
— le Sum pooling : on calcule la somme des valeurs.

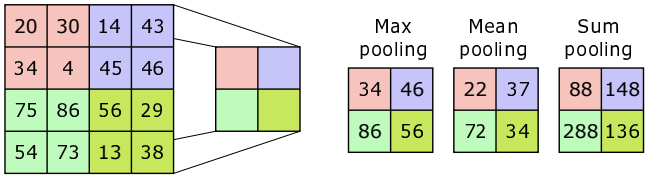


FIGURE 8.4 – Principe et variantes d'un pooling 2×2.

En pratique, le Max pooling est le fréquemment utilisé car il favorise les activations fortes. Néanmoins, les filtres de petite taille 2×2 sont préférés pour limiter la perte d'information.

8.2 Réseaux récurrents

Les réseaux de neurones récurrents sont un type de réseau possédant des connextion récurrentes, c'est-à-dire des cycles : la sortie d'un neurone est réutilisée comme une entrée de ce neurone (Figure 8.5, gauche). Ils sont particulièrement pertinents pour l'analyse de séries temporelles, en reconnaissance automatique de la parole ou de l'écriture manuscrite, et en traduction automatique.

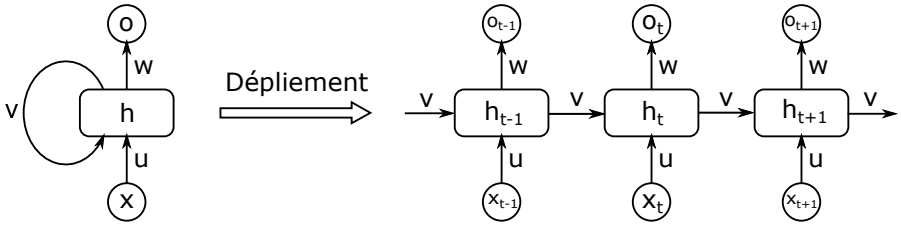


FIGURE 8.5 – Structure générale d'un réseau récurrent.

Ces réseaux sont habituellement représentés sous forme dépliée (Figure 8.5, droite). Leur structure ressemble alors à un réseau de neurones classique dont les poids entre couches seraient soumis à des contraintes d'égalité. L'intérêt de la récurrence est de fournir au réseau un effet mémoire : la sortie à l'instant $t+1$, o_{t+1}, dépend naturellement de l'entrée à l'instant $t+1$, x_{t+1}, via les poids u, mais aussi, grâce aux poids v, des entrées aux instants précédents x_t, x_{t-1}, x_{t-2}...

Les méthodes d'entraînement de ces réseaux sont les mêmes que pour les réseaux classiques, telles que la rétropropagation du gradient. Néanmoins, ils sont plus enclins aux problèmes d'évanouissement du gradient. Des architectures particulières permettent d'éviter ce problème, telles que les réseaux LSTM (*Long Short-Term Memory*), qui ne seront pas décrits plus en détails.

8.3 Réseaux auto-encodeurs

Les réseaux auto-encodeurs sont généralement considérés comme faisant partie du Deep Learning de par leur structure incluant un grand nombre de couches et certaines particularités des méthodes utilisées pour leur entrainement. Ils sont particulièrement utiles pour la réduction de dimensionnalité. Pour autant, leur structure à propagation avant ne les distingue pas des perceptrons multicouches : leur description est donc présentée en section 5.3.

8.4 Applications en Génie des Procédés

8.4.1 Classification de données

Essiet et al. (2019) utilisent un réseau convolutif à 5 couches pour classifier des données issues de capteurs, transformées sous forme d'images.

8.4.2 Chimie et synthèse organique

Coley et al. (2017) développent une structure assez complexe de 5 réseaux de neurones afin de prédire les résultats de réactions de synthèse organique (Figure 8.6). Les 4 premiers réseaux parallèles s'occupent spécifiquement d'une caractéristique de la réaction : perte et gain d'hydrogène, perte et gain de liaisons. Leurs sorties sont envoyées en entrée du cinquième réseau qui calcule le score. Les 200 000 paramètres de l'ensemble sont entraînés grâce à 5 millions de données extraites de bases de données chimiques.

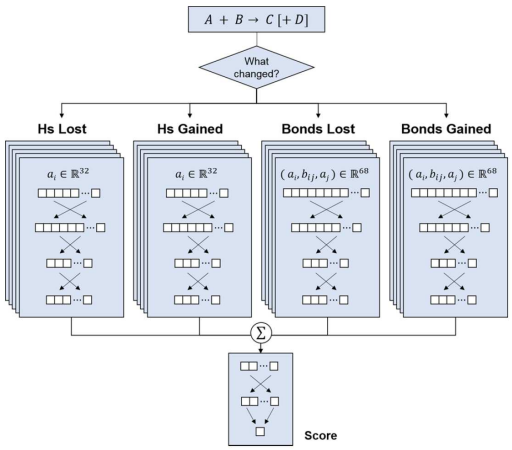


FIGURE 8.6 – Architecture des 5 réseaux de neurones entraînés pour prédire les performances de synthèses organique (Coley et al., 2017).

Nandy et al. (2018) utilisent un réseau de neurones à deux couches cachées pour découvrir de nouveaux complexes à métaux de transition.

8.4.3 Modélisation et optimisation de procédés

Himmelblau (2000) rapporte un cas de modélisation d'un réacteur de polymérisation avec un réseau récurrent interne à 4 neurones cachés.

Bollas et al. (2003) présentent deux structures de réseaux hybrides couplant un modèle phénoménologique et un réseau de neurones (pour corriger les écarts du modèle) pour la modélisation et le scale-up d'un procédéé FCC.

Loney et al. (2003) présentent plusieurs structures de réseaux de neurones très atypiques issus de méthodes évolutionnaires agissant sur les connexions et les fonctions d'activation pour différents cas d'applications.

Rosli et Aziz (2017) présentent une review des utilisations de réseaux de neurones pour modéliser le procédé de craquage : perceptrons multicouches, réseaux adaptatifs, réseaux récurrents, méthodes hybrides, etc.

Teng et al. (2019) entraînent un réseau auto-encodeur 15-25-10-2-2-13-15 (935 paramètres) grâce à 10000 données d'entrainement et 3000 données de validation, pour réduire le nombre de dimensions de 15 à 2 et faciliter l'optimisation du système. Ils utilisent une méthode évolutionnaire qui optimise le nombre de neurones par couche en plus des poids et des biais des neurones. Ils comparent leur erreur moyenne à d'autres méthodes

(ACP standard, ACP à noyaux divers, ICA, NMF) ainsi qu'au réseau auto-encodeur 15-11-7-2-7-11-15, proposé par les heuristiques classiques. Le réseau optimisé fait 20 % à 50 % d'erreurs en moins que les autres méthodes.

8.4.4 Design et synthèse de procédés

Nabil et al. (2019) présentent une nouvelle méthode de synthèse de procédés dédiée aux cycles thermodynamiques pour des installations solaires à concentration. La méthode se base sur un alphabet (chaque lettre représente une unité) permettant de construire des mots (chaque mot représente un cycle complet). L'utilisation d'un réseau de neurones récurrents permet tout d'abord d'entrainer le réseau à comprendre cette langue des cycles. Le réseau est ensuite exploité pour générer de nouveaux cycles innovants à partir de cet alphabet, qui sont simulés et validés ou rejetés. Puis le réseau est forcé à générer des cycles présentant des caractéristiques énergétiques améliorées. Cette méthode permet de retrouver les cycles optimisés de la littérature et des cycles encore inconnus présentant des performances optimales.

8.4.5 Rectification et réconciliation de données

Himmelblau (2000) montre un exemple de rectification de données d'une cuve agitée grâce à un réseau récurrent.

8.4.6 Détection et diagnostic de fautes et dérives

Venkatasubramanian et al. (1989) sont parmis les premiers à étudier les caractéristiques de réseaux de neurones multicouches pour la classification de données de procédés en vue de la détection d'anomalies de fonctionnement. Ils comparent plusieurs réseaux (perceptrons monocouches de 18-5-13 à 18-27-13 et multicouches jusqu'à 18-27-27-13) pour relier les 18 mesures issues d'un procédé aux 13 types de défauts observés. Leur comparaison montre que les perceptrons monocouches sont plus efficaces que les multicouches.

8.4.7 Contrôle, commande et régulation de procédés

Himmelblau (2000) rapporte un cas de contrôle de procédé par réseau récurrent.

Santos et al. (2013) entraînent un réseau récurrent d'Elman 3-3-1 à 25 paramètres pour construire un modèle en vue de la régulation d'un procédé batch de polymérisation de styrène.

Wong et al. (2018) utilisent un réseau de neurones récurrent pour construire le modèle d'un procédé continu de synthèse pharmaceutique afin de l'exploiter dans le cadre d'une commande prédictive basée sur le modèle.

Chapitre 9

Méthodes en flux

Dans beaucoup d'applications, les données à traiter pour l'analyse ou la construction de modèles ne sont pas disponibles hors-ligne mais sont générées en temps réel. Il peut alors être plus pertinent de les traiter "à la volée" dès qu'elles sont disponibles, plutôt que de les stocker pour les retraiter ensuite. Le problème du traitement temps-réel est très différent selon qu'il s'agit d'analyse de ces données ou de leur modélisation.

Dans le cas d'une analyse de données, l'analyse peut être effectuée en utilisant un modèle pré-entrainé : chaque nouvelle donnée est présentée au modèle, et le calcul de la réponse peut être très rapide. Par exemple, si l'analyse des données issues des capteurs d'un procédé vise à détecter d'éventuelles dérives, le jeu de mesures des capteurs à un instant t peut être soumis en entrée d'un réseau de neurones pré-entrainé, qui fournira quasi-instantanément une réponse permettant de savoir si une dérie est en cours ou non.

Si l'objectif du traitement de données vise, par contre, à affiner un modèle ou entrainer une méthode, alors cette étape peut être beaucoup plus longue en temps de calcul et donc ne pas être compatible avec la fréquence de génération des nouvelles données. Le ré-entrainement complet du modèle, sur la base des données historiques et de la nouvelle donnée, n'est pas possible dans un temps restreint. Des stratégies de stockage partiel et de traitement en batch existent mais ne seront pas détaillées ici. On se focalisera sur des variantes spécifiques de méthodes permettant d'actualiser un modèle pour intégrer la nouvelle informations reçue.

9.1 Construction et actualisation de modèles

9.1.1 Régression multilinéaire récursive

La régression multilinéaire classique peut facilement s'appliquer sur un lot de données historiques pour corréler les sorties expérimentales $y^{(t)}$ (où t désigne l'indice de pas de temps temporel plutôt que l'instant lui-même : $0 \leq t \leq n$) en fonction des variables $x_j^{(t)}$ $(1 \leq j \leq p)$ selon un modèle mathématique de forme :

$$y_{mod}^{(t)} = f(x^{(t)}; a) = \sum_{k=1}^{m} a_k \phi_k(x^{(t)}) = \Phi(x^{(t)}).a \tag{9.1}$$

où le régresseur Φ peut être écrit :

$$\Phi = \begin{pmatrix} \phi_1(x^{(0)}) & \phi_2(x^{(0)}) & \cdots & \phi_m(x^{(0)}) \\ \phi_1(x^{(1)}) & \phi_2(x^{(1)}) & \cdots & \phi_m(x^{(1)}) \\ \phi_1(x^{(2)}) & \phi_2(x^{(2)}) & \cdots & \phi_m(x^{(2)}) \\ \vdots & \vdots & & \vdots \\ \phi_1(x^{(n)}) & \phi_2(x^{(n)}) & \cdots & \phi_m(x^{(n)}) \end{pmatrix} \tag{9.2}$$

Le vecteur des m paramètres du modèle, a, i.e. les coefficients inconnus, peut alors être estimé tel que (voir section 2.2) :

$$\hat{a} = (\Phi^T \Phi)^{-1} \Phi^T y \tag{9.3}$$

Si on souhaite appliquer cette même méthode à chaque fois qu'une nouvelle donnée $(x^{(t+1)}, y^{(t+1)})$ est disponible, il semble nécessaire de rajouter une ligne à la matrice Φ et de recalculer $\hat{a}$. Comme chaque nouvelle donnée augmente la taile de Φ, chaque nouvelle donnée demandera plus de stockage et de temps de calcul, et finira par rendre cette logique inappropriée.

Il existe néanmoins une variante récursive de la régression multilinéaire qui permet de ne pas refaire tout le calcul, mais de simplement réactualiser les coefficients $\hat{a}$ grâce à chaque nouvelle donnée (Walter et Pronzato, 1994).

Pour mettre en place cette méthode, on distingue les caractéristiques du modèle par leur instant t :

— A chaque instant t, une donnée $(x^{(t)}, y^{(t)})$ est disponible.
— Le régresseur partiel r_t^T correspond à une ligne de la matrice Φ_t,
— La matrice Φ_t prend en compte tous les termes du modèle mathématique sur toutes les données entre l'instant 0 et l'instant t. Elle est nécessaire pour les premières données mais n'aura pas besoin d'être conservée à partir d'un nombre de données suffisant.
— On notera $\hat{a_t}$ l'estimation des paramètres du modèle obtenue à l'instant t.
— On introduit également une nouvelle matrice P_t qui sera réactualisée avec chaque nouvelle donnée.

Au début de l'application de la méthode, on ne dispose pas encore d'assez de données pour effectuer une régression multilinéaire. Il faut attendre d'avoir au moins m données pour démarrer, où m est le nombre de paramètres inconnus du modèle. Dès qu'on atteint un temps t qui a permis d'accumuler m données, on initialise la récurrence :

— On calcule alors Φ_t,
— On calcule la première estimation des coefficients du modèle $\hat{a_t} = (\Phi_t^T \Phi_t)^{-1} \Phi_t^T y_t$ sur la base des données disponibles,
— On initialise la matrice $P_t = (\Phi_t^T \Phi_t)^{-1}$.

A l'instant $t+1$, une nouvelle donnée est disponible $(x^{(t+1)}, y^{(t+1)})$. On peut alors réactualiser ces grandeurs pour faire évoluer le modèle :

— On calcule le régresseur r_{t+1} à partir de $x^{(t+1)}$,
— On calcule le gain de correction :

$$k_{t+1} = \frac{P_t r_{t+1}}{1 + r_{t+1}^T P_t r_{t+1}} \tag{9.4}$$

— On actualise la matrice P :

$$P_{t+1} = P_t - k_{t+1} r_{t+1}^T P_t \tag{9.5}$$

— On peut alors actualiser les coefficients du modèle :

$$\hat{a}_{t+1} = \hat{a}_t + k_{t+1}\big(y^{(t+1)} - r_{t+1}^T . a_t\big) \tag{9.6}$$

Chapitre 10

Outils logiciels

Depuis les années 2010 et l'explosion des succès d'application des méthodes d'intelligence artificielle, notamment grâce à l'apprentissage profond, le nombre de logiciels et d'environnements de programmation dédiés à ces méthodes s'est également décuplé. Cet élan a été porté par le fait que les principaux développements ont été diffusés en accès libre, permettant leur incorporation rapide dans ces environnements de programmation par les développeurs eux-mêmes. La communauté s'est également structurée et organisée pour utiliser des standards avec l'impulsion forte de GAFAM tels que Google, Apple et Facebook, dont les départements d'intelligence artificielle sont très actifs.

On présentera ci-dessous rapidement les principaux environnements disponibles, puis on se focalisera sur l'environnement MATLAB, qui n'est pas le plus puissant de tous, mais permet une prise en main rapide et une utilisation conviviale.

10.1 Outils logiciels

Cette section ne mentionne que rapidement les pricipaux environnements de développement. Le lecteur intéressé est invité à consulter les références suivantes pour une vue plus détaillée : Espinasse et Bellot (2017), Yao et Ge (2018), Tufféry (2019), Zhou et al. (2020).

Scikit Learn Python - Le langage Python s'est imposé en apprantissage automatique depar la diversité des objets qu'il est capable de manipuler. Une large communauté s'est fédérée et a permis de développer de nombreux modules fonctionnels pour implémenter les principales méthodes de fouille de données et d'apprentissage, notamment via le projet Scikit-Learn. On peut citer les librairies Annoy (recherche des plus proches voisins), Caffe (Deep learning framework), Chainer (réseaux de neurones), neon (Deep Learning), NuPIC, Shogun (Machine Learning Toolbox), TensorFlow (Réseau de neurones disposant d'une API de haut niveau), Theano (apprentissage automatique pour des expressions mathématiques), Torch (Framework d'algorithmes d'apprentissage), Theanets (deep learning).

Tensorflow - TensorFlow est un outil open source d'apprentissage automatique développé par Google, ouvert en 2015 et publié sous licence Apache. Il est fondé sur l'infrastructure DistBelief et est doté d'une interface pour Python, Julia et R. C'est l'un des outils les plus utilisés en apprentissage machine. TensorFlow regroupe un grand nombre de modèles et d'algorithmes de Machine Learning et de Deep Learning. Il permet notamment d'entraîner et d'exécuter des réseaux de neurones pour la classification, la reconnaissance d'image, les plongements de mots, les réseaux de neurones récurrents, les modèles sequence-to-sequence pour la traduction automatique, ou encore le traitement naturel du langage.

R - R est un logiciel généraliste de traitement statistique de données, largement utilisé par les statisticiens, les data miners, data scientists pour le développement de logiciels statistiques et l'analyse des données. Il fonctionne sous la forme d'un interpréteur de commandes. Il dispose d'une bibliothèque très large de fonctions statistiques, et il est possible d'en intégrer de nouvelles par des "packages", des modules externes compilés (sous forme de DLL sous Windows). Il est possible de l'utiliser en mode interfacé sans avoir à programmer.

Weka - Weka (Waikato Environment for Knowledge Analysis,Environnement Waikato pour l'analyse de connaissances) est une suite de logiciels d'apprentissage automatique écrite en Java et développée à l'université de Waikato en Nouvelle-Zélande. Weka est un logiciel libre disponible sous la Licence publique générale GNU (GPL).

10.2 Matlab

Toute cette section est dédiée à l'utilisation de Matlab pour la mise en oeuvre de méthodes d'apprentissage automatique, soit par le biais de leur programmation, soit par l'utilisation des fonctions prédéfinies dans Matlab, notamment les toolbox dédiées aux réseaux de neurones, au Machine Learning et au Deep Learning. Les différentes sections suivent globalement le plan de ce support de cours. Il ne s'agit pas d'un mode d'emploi détaillé de Matlab, ni d'un guide de prise en main, on supposera que le lecteur possède déjà une expérience minimale de l'environnement Matlab.

Pour aider à la prise en main de ces fonctions, l'aide en ligne de Matlab est très utile car très bien documentée, soit depuis l'interface Matlab dans l'onglet Help, soit sur leur site web où de nombreuses vidéos et tutoriels sont disponibles. On rappelle au lecteur qu'il est possible, dans l'interface Matlab, d'obtenir une aide textuelle rapide sur une fonction `lafonction`, en tapant simplement dans la fenêtre de commande `help lafonction`.

Dans la suite, les notations génériques suivantes seront utilisées :

— s, s_i, x : désignent des scalaires,
— i, j, m, n, p, r : désignent des entiers,
— V, V_n : désignent des vecteurs,
— M, $M_{m,n}$: désignent des matrices ou tableaux,
— t : désigne une chaîne de caractères,
— ... : signifie que la syntaxe peut difficilement être résumée : il est préférable de se référer à la documentation en ligne.

10.2.1 Fonctions et instructions généralistes

Le Tableau 10.2 rappelle les principales fonctions généralistes utiles dans la fenêtre de commande Matlab ou dans des scripts.

Fonction	Description
`clc`	Nettoie la fenêtre de commandes
`clear all`	Réinitialise le workspace en supprimant toutes les variables en mémoire
`help nomfonction`	Affiche dans la fenêtre de commande une aide simplifiée sur la fonction `nomfonction`
%	Le symbole % permet de définir un commentaire
%%	Le symbole %% dans un script Matlab permet de créer une "section" executable sans executer le reste du script
;	Le point-virgule à la fin d'une ligne de script empêche l'affichage des "sorties" de cette ligne

TABLE 10.1 – Fonctions Matlab généralistes.

10.2.2 Variables, vecteurs, tableaux

Les principales créations et manipulations élémentaires de variables sont listées dans le Tableau 10.2.

10.2.3 Fonctions mathématiques et statistiques

Les principales fonctions mathématiques et statistiques utiles sont listées dans le Tableau 10.3.

10.2.4 Fonctions graphiques

Les principales fonctions graphiques utiles sont listées dans le Tableau 10.4.

10.2.5 Structures algorithmiques de base

Les boucles indicées s'écrivent :

```
for r = 1:n
    for c = 1:n
        A(r,c) = 1/(r+c-1);
    end
end
```

Fonction	Description
`M=eye(n)`	Crée une matrice identité $n \times n$
`M=ones(m,n)`	Crée une matrice remplie de 1 de taille $m \times n$
`M=zeros(m,n)`	Crée une matrice remplie de 0 de taille $m \times n$
`inv(M)`	Inverse d'une matrice
`transpose(M)` ou `M'`	Transposée d'une matrice
`[1 2 3 4 5]`	Crée un vecteur ligne [1 2 3 4 5]
`[1; 2; 3; 4; 5]`	Crée un vecteur colonne $[1\ 2\ 3\ 4\ 5]^T$
`indices = find(V == a)`	Permet de trouver les `indices` des éléments de `V` qui sont égaux au scalaire `a`. La condition `V == a` peut être remplacée par tout test logique.
`length(V)`	Fournit la longueur (sa taille) du vecteur V
`V=linspace(a,b,n)`	Crée un vecteur de n éléments uniformément répartis entre a et b
`size(M)`	Fournit les dimensions du tableau M
`size(M,k)`	Fournit la longueur de la k^{ieme} dimension du tableau M
`V=logspace(a,b,n)`	Crée un vecteur de n éléments répartis selon une progression géométrique entre 10^a et 10^b
`V(:)`	Désigne tous les éléments du vecteur V
`V(i)`	Désigne l'élément i du vecteur V
`U=V(j:k)`	Stocke dans U les éléments de V d'incides compris entre j et k
`[ ]`	Crée un vecteur/tableau vide qui pourra ensuite être étendu par concaténation
`'blablabla'`	Crée une chaîne de caractères
`C={1, 'bla', [0 1]}`	Les tableaux de cellules (cell array) sont un type de variables qui permet de stocker des données de types différents. Leur construction se fait par accolades.
`T = table(var1,...,varN)`	Les tables sont un type de variables qui permet de stocker des types de données différentes sous forme de table : chaque colonne est de type uniforme.

TABLE 10.2 – Variables, vecteurs, tableaux.

Fonction	Description
$M_{p,p}$=`corrcoef(`$M_{n,p}$`)`	Matrice de coefficients de corrélation des colonnes de $M_{n,p}$
$M_{p,p}$=`cov(`$M_{n,p}$`)`	Matrice de covariance des colonnes de $M_{n,p}$
`[V`$_{n,n}$`,D`$_{n,n}$`] = eig(M`$_{n,n}$`)`	Stocke dans les colonnes de V les vecteurs propres, et sur la diagonale de D les valeurs propres de la matrice M
`log(x)`	Logarithme népérien d'un scalaire
`log10(x)`	Logarithme décimal d'un scalaire
`s=mean(V)`	Moyenne d'un vecteur
`V=mean(M,n)`	Vecteur des moyennes d'une matrice selon les lignes ou les colonnes selon la valeur de n
`s=mean(mean(M))`	Moyenne d'une matrice
`s=min(V)`	Minimum d'un vecteur
`s=min(min(M))`	Minimum d'une matrice
`s=max(V)`	Maximum d'un vecteur
`s=max(max(M))`	Maximum d'une matrice
`V=pdist(`$M_{n,p}$`)`	Calcule et stocke dans V les distances deux-à-deux de n données stockées dans M dans un espace à p dimensions
`V=pdist2(`$M_{1,n_1,p}$`,`$M_{2,n_2,p}$`)`	Stocke dans V les distances deux-à-deux entre les n_1 données de M_1 et les n_2 données de M_2 dans un espace à p dimensions. Utile pour détecter les plus proches voisins
`s=rand()`	Génère un réel aléatoire s entre 0 et 1
`r=randn(x,s)`	Génère un réel aléatoire r selon une distribution gaussienne de moyenne x et d'écart-type s
`s=std(V)`	Ecart-type des données d'un vecteur
`s=sum(V)`	Somme des éléments d'un vecteur
`s=sum(sum(M))`	Somme des éléments d'une matrice
`s=tinv(alpha,nu)`	Retour l'opposé de la variable de Student $t_{\alpha,\nu}$, c'est-à-dire $-t_{\alpha,\nu}$
`s=var(V)`	Variance des éléments d'un vecteur
`V.`$\hat{2}$	Met au carré chacun des éléments du vecteur V
`[`$M_{n,p}$ $M_{n,q}$`]`	Concatène horizontalement les deux matrices qui ont le même nombre de lignes n
`[`$M_{m,p}$`;` $M_{n,p}$`]`	Concatène verticalement les deux matrices qui ont le même nombre de colonnes p
$M_{n,p+1}$ `= [`$M_{n,p}$ $V_{n,1}$`]`	Rajoute une colonne à la matrice M en concaténant un vecteur à droite

TABLE 10.3 – Fonctions mathématiques et statistiques.

Fonction	Description
`axis([`x_{min} x_{max} y_{min} y_{max}`])`	Contraint les axes x et y d'une figure $y = f(x)$ entre les bornes x_{min}, x_{max}, y_{min} et y_{max}
`bar(Vn)`	Trace les valeurs du vecteur V_n sous forme d'un diagramme de barres verticales
`clf`	Efface le contenu d'une figure
`contour(Vx,Vy,Mxy,...)`	Trace une carte iso-contour des données de la matrice Mxy d'abscisses Vx et d'ordonnées Vy
`contourf(Vx,Vy,Mxy,...)`	Fonctionne comme `contour`, mais colorie les surfaces entre courbes iso-valeurs
`figure(n)`	Crée la fenêtre graphique de la figure n
`gplotmatrix(Mnp,Mnq,Vn)`	Trace une matrice de projections et distributions des colonnes de Mnp en fonction des colonnes de Mnp. Chaque donnée est colorée selon les groupes d'indices dans Vn
`gscatter(Vx,Vy,Vi)`	Trace les points de coordonnées $(Vx; Vy)$ en différenciant leurs couleurs en fonction du vecteur d'indices Vi : les points ayant la même valeur dans Vi sont tracés de la même couleur.
`histogram(V,n,...)`	Crée et trace un histogramme des valeurs contenues dans le vecteur V en les classant dans n classes, avec options possibles.
`hold on`	Permet de superposer un tracé sur une figure
`image(M)`	Trace une image de la matrice M en colorant les cases en fonction de ses valeurs. Pour aller plus vite quelles que soient les valeurs de M : `image(M,'CDataMapping','scaled')`
`legend(...)`	Permet d'inclure une légende
`plot(Vx,Vy,...)`	Trace les données de Vy en fonction de leurs abscisses Vx
`subplot(m,n,i)`	Active la sous-figure i d'une matrice de figures à m lignes et n colonnes
`title('t')`	Affiche le titre t au-dessus de la figure
`xlabel('t')`	Affiche le titre t sur l'axe des abscisses
`ylabel('t')`	Affiche le titre t sur l'axe des ordonnées

TABLE 10.4 – Fonctions graphiques.

Les tests conditionnels s'écrivent :

```
if i == i
    A(i,j) = 2;
elseif abs(i-j) == 1
    A(i,j) = -1;
else
    A(i,j) = 0;
end
```

10.2.6 Actions sur des fichiers

`data=load('monfichier.txt')` : permet de lire rapidement un fichier de données numériques stockées sous forme de colonnes à largeurs fixes. L'ensemble est stocké dans la matrice `data`.

`data = readtable('parc-regional-annuel-prod-eolien-solaire_mini2.csv')` : permet de lire des fichiers de données CSV (comma separated values) contenant des donnnées de types divers séparées par des virgules ou des point-virgules. Le résultat est stocké dans la table data.

`save('messauvegardes.mat',variables)` : permet de sauver les variables variables dans un fichier Matlab 'messauvegardes.mat'. Ce n'est pas un fichier texte, il ne peut être ouvert que depuis Matlab.

L'export de variables vers des fichiers texte est plus compliquée que leur lecture avec load. Il faut ouvrir le fichier, écrire dedans avec une format spécifié, et le fermer :

```
fileID = fopen('monfichier.txt');       % ouverture
fprintf(fileID,'%f %f\n',data(:,1:2);   % 2 réels par ligne et retour à la ligne
fclose(fileID);                         % fermeture
```

10.2.7 Clustering et partitionnement

K-moyennes

La méthode des K-moyennes s'applique en appelant :
`[idx,C,sumd,D] = kmeans(data,K)`
qui reçoit un tableau `data` à n lignes et p colonnes, et le nombre K de clusters à identifier. La fonction renvoie :

- `idx` : un vecteur $n \times 1$ qui contient les indices d'affectation des n données à l'un des K clusters,
- `C` : un tableau $K \times p$ des coordonnées des centroïdes des clusters,
- `sumd` : un vecteur $K \times 1$ des distances intra-clusters,
- `D` : un tableau $n \times K$ des distances de chaque données i à chacun des centroïdes des clusters.

Clustering hiérarchique et dendrogramme

Pour calculer et tracer un dendrogramme de clustering hiérarchique (voir section 4.4 et exemple de la Figure 10.1), Matlab a besoin :

- de faire appel à la fonction `pdist` pour calculer la distribution `distrib` des distances (euclidienne ou autre) entre données deux-à-deux à partir des données contenues en lignes dans `data`,
- de créer la structure de dendrogramme, par la fonction `linkage`, en appliquant la méthode de clustering hiérarchique,
- de tracer ce dendrogramme dans une fenêtre de figure avec la fonction `dendrogram` :

```
distrib = pdist(data,'euclidean');
clustTree = linkage(distrib,'average');
dendrogram(clustTree,0);
```

DBSCAN

La méthode DBSCAN est très simple à utiliser dans Matlab. Elle requiert de définir tout d'abord les deux paramètres permettant de définir un voisinage : `epsilon` et `minpts`. Le paramètre `epsilon` définit le rayon du voisinage dans lequel la méthode va compter le nombre de voisins pour savoir si une donnée est isolée ou au sein d'une zone dense. Le seuil de densité est défini par `minpts`. Ces deux paramètres sont fournis à la fonction `dbscan`, ainsi que la matrice des données `data`, elle renvoit les indices des clusters de chaque donnée, `indicesclusters`. On peut alors très rapidement tracer une projection de ces clusters avec la fonction `gscatter` qui applique des symboles identiques aux données possédant les mêmes valeurs dans `indicesclusters`.

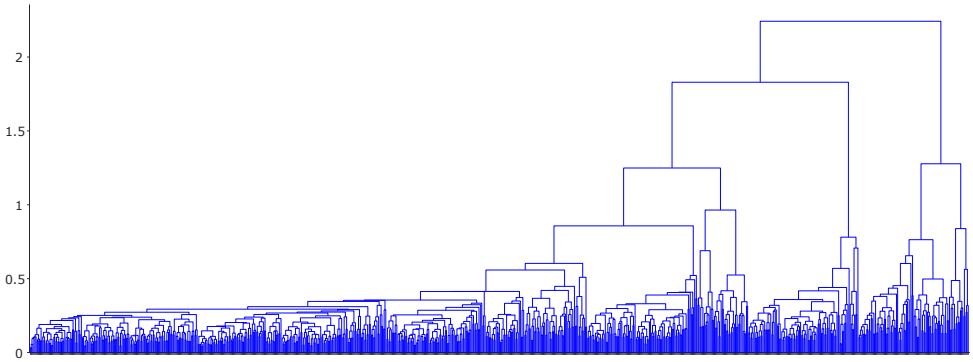


FIGURE 10.1 – Exemple de dendrogramme généré par Matlab à partir de 841 données extraites d'un procédé de carbonatation.

```
epsilon = 1;
minpts = 5;
indicesclusters = dbscan(data,epsilon,minpts);
gscatter(data(:,1),data(:,2),indicesclusters);
```

Le vecteur `indicesclusters` contient les indices de clusters des données : 1, 2, 3, ... pour les données appartenant à des clusters denses, et -1 pour les données isolées. Le nombre de clusters est donc donné par `max(indicesclusters)`, et le nombre de points isolés par le nombre de -1 présents dans ce vecteur.

Carte auto-adaptative de Kohonen

Les cartes auto-adaptatives de Kohonen étant une forme particulière de réseau de neurones, elles peuvent soit être construites dans un script, ce qui est assez délicat pour débuter, soit être mises en oeuvre grâce à l'outil "Neural Networks" de Matlab. Pour illustrer son application, on utilisera un ensemble de 600 données dans $\mathbb{R}^2$ représentées sur la Figure 10.2 et qui contiennent clairement 2 clusters.

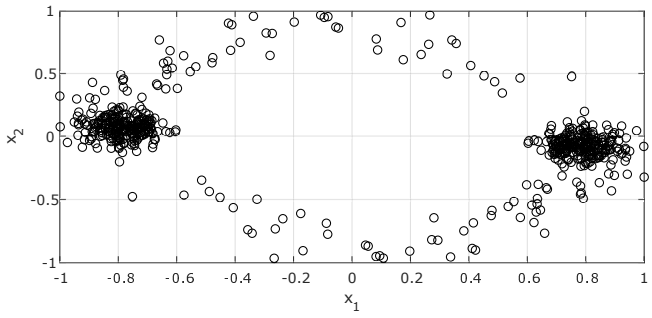


FIGURE 10.2 – Données utilisées pour un clustering par cartes auto-adaptatives de Kohonen.

La première étape pour construire une carte auto-adaptative consiste à lancer l'outil "Neural Networks" de Matlab. Il suffit de taper dans la fenêtre de commande : `nnstart`. Cela ouvre alors la fenêtre d'accueil de l'outil (Figure 10.3). Il faut alors cliquer sur le "Clustering app".

Après une nouvelle fenêtre d'accueil et un clic sur Next, on aboutit à la page de sélection des données (Figure 10.4). On peut alors sélectionner les données à partitionner, soit en les sélectionnant directement avec le menu déroulant qui donne accès aux variables du workspace, soit avec le bouton "..." qui permet de lire des fichiers. Il faut alors bien préciser si les données sont en lignes ou en colonnes : Matlab les lit en colonnes par défaut, alors qu'elle sont plus couramment présentées en lignes dans les tableaux de données.

La fenêtre "Network Architecture" permet de fixer le nombre de neurones pour la carte (Figure 10.5). Par défaut, la valeur vaut 10, ce qui signifie que la carte sera de taille 10×10. Elle aura également une trame hexagonale par défaut. Puisque notre exemple contient 2 entrées et 100 neurones, il y a donc 200 poids à entrainer. Comme le jeu de données contient 600 données, cela semble raisonnable, mais on aurait pu réduire la taille de la carte.

La fenêtre suivante est celle d'entrainement (Figure 10.6). Initialement, seul le bouton "Train" est actif, il provoque l'entrainement du réseau. Ensuite, les 4 boutons à droite deviennent accessibles, ils permettent de voir les résultats :

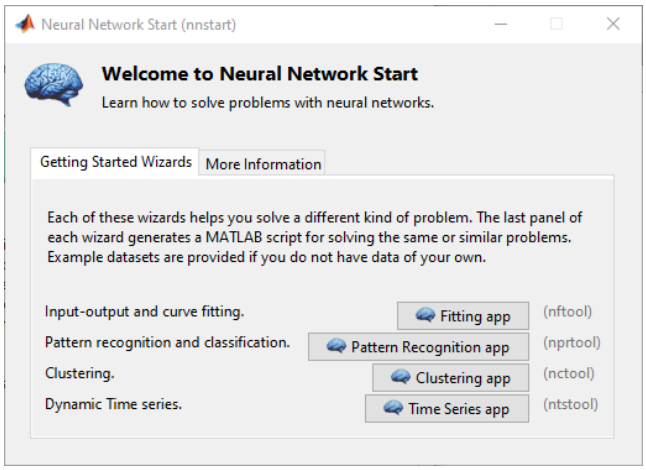


FIGURE 10.3 – Fenêtre d'accueil de l'outil "Neural Network".

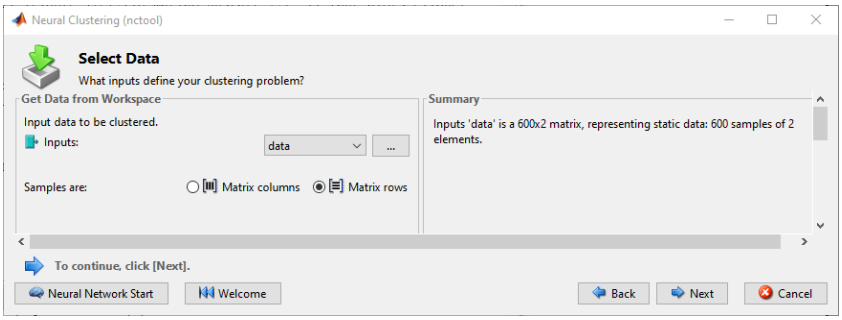


FIGURE 10.4 – Fenêtre de sélection des données de l'outil "Neural Clustering".

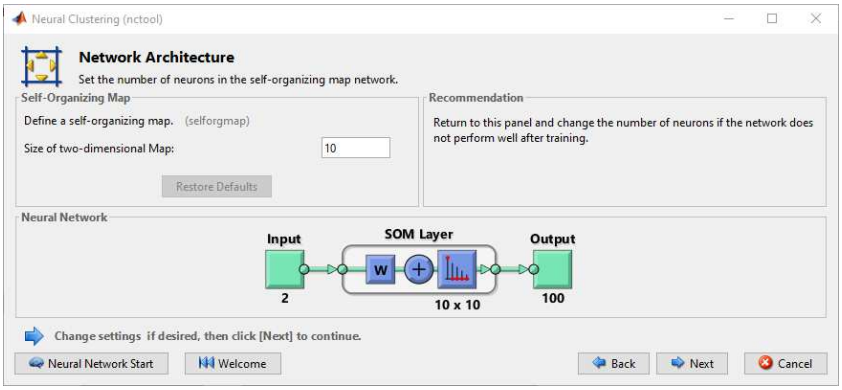


FIGURE 10.5 – Fenêtre de sélection du nombre de neurones de la carte de Kohonen.

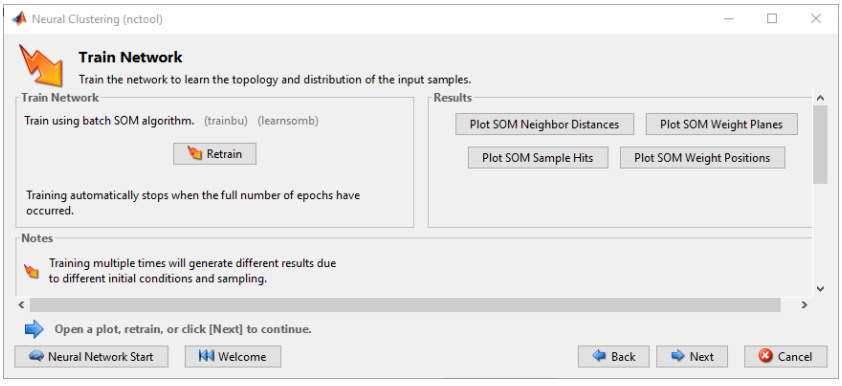


FIGURE 10.6 – Fenêtre d'entrainement et d'affichage des résultats.

— Plot SOM Neighbour Distances (Figure 10.7 (a)) : cette carte présente les distances entre neurones voisins de la carte. Chaque neurone est symbolisé par un hexagone gris : les liens d'un neurone avec ses voisins sont colorés en jaune lorsque la distance est petite, donc lorsque les deux neurones se ressemblent, et en couleur sombre lorsque la distance est élevée, donc lorsque les deux neurones ne se ressemblent pas. Les cases sombres indiquent les frontières entre clusters. On observe ici une zone sombre qui coupe la carte en deux, indiquant qu'on peut identifier deux clusters dans ces données.
— Plot SOM Weight Planes (Figure 10.7 (b)) : Ces 2 cartes représentent les poids du réseau. Puisque chaque entrée est reliée à chaque neurone, on a donc 100 poids pour l'entrée x_1 et 100 poids pour l'entrée x_2. Ces deux cartes visualisent les valeurs de ces 100 + 100 poids.
— Plot SOM Sample Hits (Figure 10.7 (c)) : Cette carte indique pour chaque neurone le nombre de données pour lesquelles ce neurone est le "neurone gagnant", c'est-à-dire celui auquel elles ressemblent le plus. On voit clairement deux paquets de données qui se regroupent à droite et à gauche, correspondant aux deux clusters. On observe également des cases vides sur cette carte, ce sont des neurones qui ne sont proches d'aucune donnée : on les repère bien sur la Figure (d) car ils se retrouvent au centre du cercle que constituent les données.
— Plot SOM Weight Positions (Figure 10.7 (d)) : Puisque chaque neurone est de même dimension qu'une donnée, on peut également les représenter dans l'espace des données. Cette carte superpose les données et les neurones. La carte de Kohonen peut être interprétée comme un filet déformable qu'on a étiré pour qu'il colle aux données. On voit bien que l'essentiel des noeuds de ce filet sont regroupés autour des deux clusters, mais qu'il existe aussi des noeuds qui s'étendent le long des deux "bras" qui relient ces deux clusters.

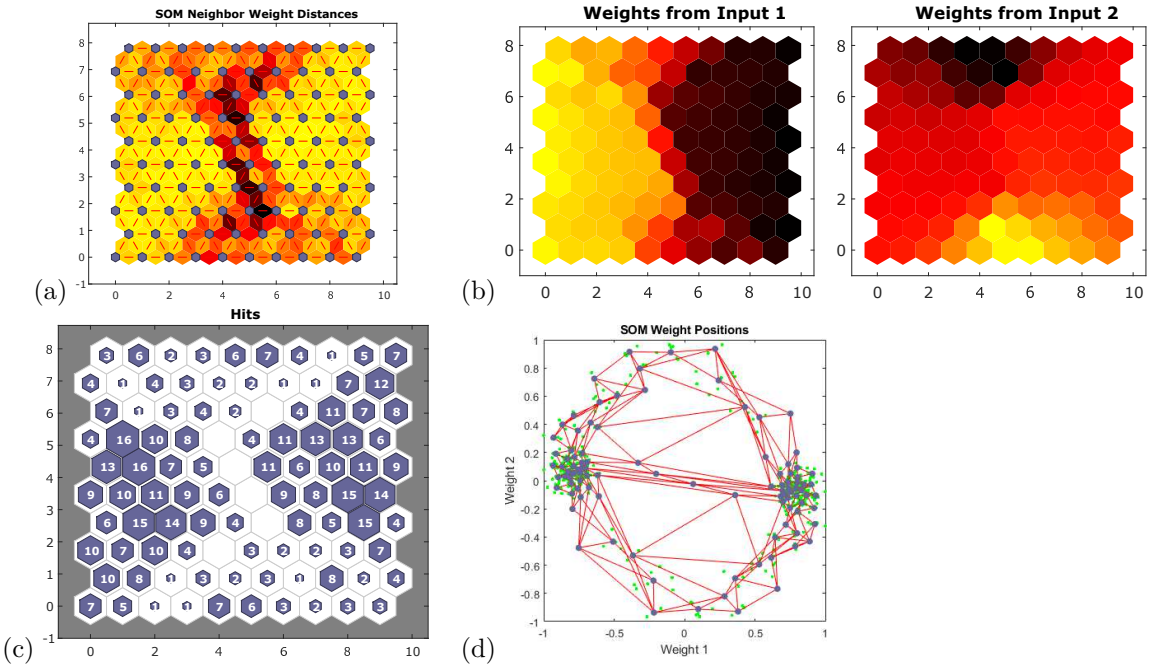


FIGURE 10.7 – Résultats de la création d'une carte auto-adaptative.

La fenêtre suivante (non présentée) permet (i) de ré-entrainer le réseau en le réinitialisant, (ii) de recommencer en modifiant la taille du réseau, (iii) d'importer un jeu de données de plus rgande taille, ou (iv) de réaliser des tests sur de nouvelles variables. Cette dernière possibilité peut être exploitée pour de la classification (voir section 10.2.9).

La fenêtre de déploiement de solutions (Figure 10.8) permet de (i) générer des fonctions Matlab pour réutiliser la carte, (ii) d'exporter la carte dans un objet Simulink ou (iii) d'exporter une image du diagramme. Enfin, la fenêtre de sauvegarde (Figure 10.9) permet (i) de générer des scripts Matlab, simple ou "advanced", qui permettront de refaire exactement le même type d'entrainement sans passer par l'interface, ou (ii) d'exporter le réseau et ses sorties directement dans le workspace sous forme de structures ou de tableaux.

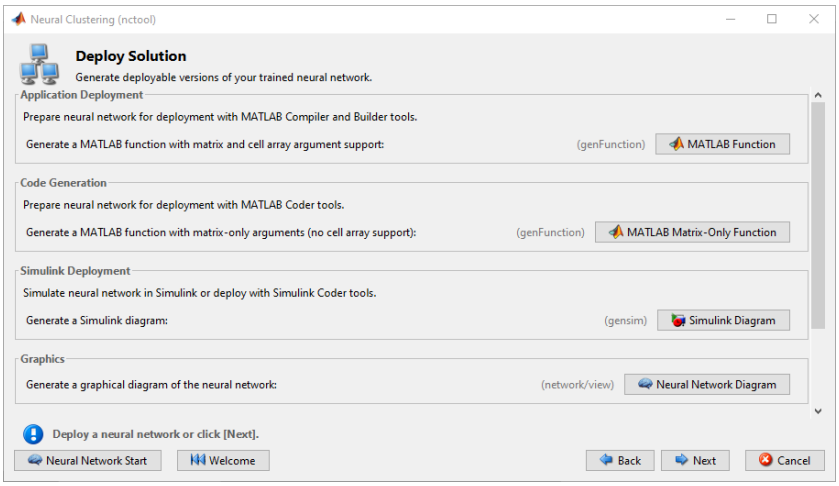


FIGURE 10.8 – Fenêtre de génération de versions d'export.

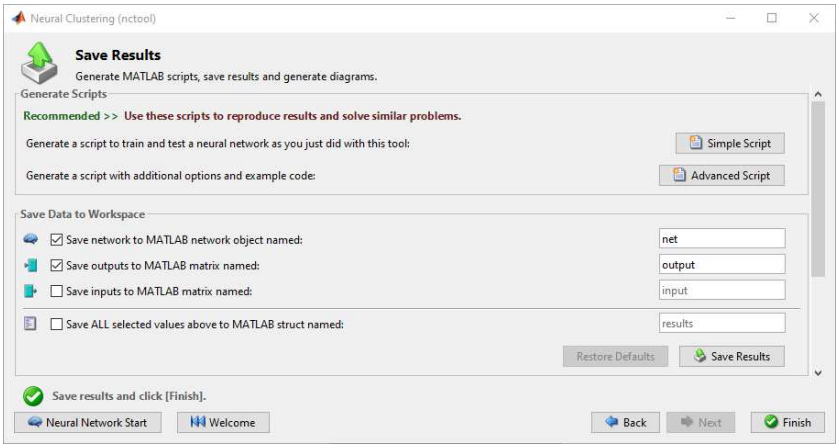


FIGURE 10.9 – Fenêtre de sauvegarde des résultats de la carte auto-adaptatives.

10.2.8 Réduction de dimensionnalité

Analyse en Composantes Principales

L'analyse en composantes principales, détaillée en section 5.1, peut facilement être appliquée sur des données normalisées en faisant appel à la fonction `pca` sous la forme :

```
[composantes,projections,variances,tsquared,fracvariances,mu] = pca(data)
```

Le tableau `data` ($n \times p$), fourni en argument, contient les données en lignes, chaque colonne correspondant à une variable : on considère n données dans $\mathbb{R}^p$. Les variables renvoyées par `pca` sont :

- `composantes` : ce tableau $p \times p$ fournit, en colonnes, les vecteurs représentant les composantes principales, donc la nouvelle base de vecteurs qui va permettre de réduire la dimensionnalité du problème. La sléection du nombre de composantes à retenir n'est pas automatique, il devra être fait par l'utilisateur en prenant connaissance du tableau `fracvariances`.
- `projections` : ce tableau $n \times p$ fournit les nouvelles coordonnées des données dans la base des composantes principales, il s'agit donc des projections des coordonnées originales $(x_1; x_2)$ dans cette base.
- `variances` : ce vecteur de taille p fournit les variances des données selon chaque composante principale.
- `tsquared` : le vecteur `tsquared` de taille n fournit, pour chaque donnée, la variable t^2 de Hotelling qui est une généralisation maulti-vériée de la variable t de Student. C'est une mesure de la distance qui sépare chaque donnée du centroïde des données.
- `fracvariances` : ce vecteur de taille p fournit la fraction de la variance totale expliquée par chacune des composantes principales. C'est grâce à ce vecteur qu'on peut sélectionner le nombre de composantes principales à retenir pour effectuer la réduction de dimensionnalité sans perdre trop d'information.
- `mu` : ce vecteur de taille p fournit les moyennes des variables $x_j (1 \leq j \leq p)$ originales.

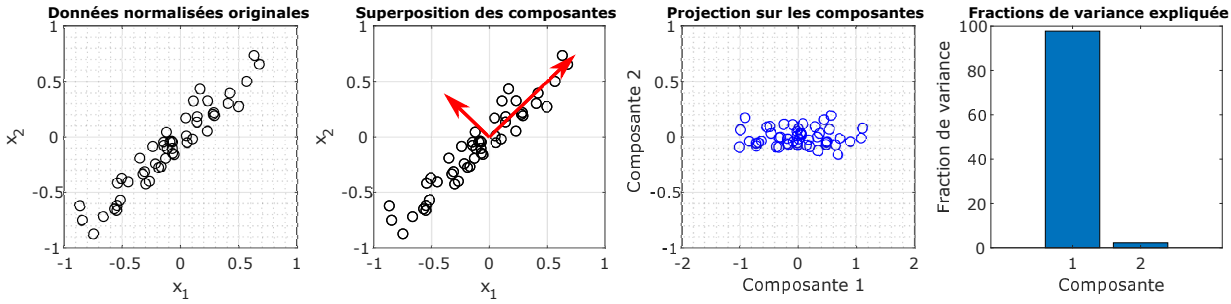


FIGURE 10.10 – Illustration des données et des résultats d'une ACP.

La Figure 10.10 présente les données originales auxquelles l'analyse en composantes principales a été appliquée (gauche) par un appel à la fonction `pca`. Le tableau `composantes` a permis de tracer sur la deuxième figure les deux composantes principales détectées, en rouge. La troisième figure représente les données dans la base des deux composantes principales. La quatrième figure montre les fractions de variance expliquée par chacune des composantes principales : on constate que la première composante explique à elle seule 97 % de la variance totale. On peut donc tout à fait réduire la dimensionnalité du problème à 1 en se limitant à cette composante principale comme unique variable permettant de représenter les variables.

Cartes auto-adaptatives

L'utilisation des cartes auto-adaptatives est présentée dans la section 10.2.7.

Réseaux auto-encodeurs

Puisque les réseaux auto-encodeurs relèvent plutôt des réseaux complexes, leur mise en oeuvre est présentée en section 10.2.11.

10.2.9 Classification

Classification Learner

Alors que la plupart des méthodes de classification peuvent être mises en oeuvre dans Matlab dans des cripts, il existe une application très simple d'utilisation qui permet de toutes les tester et de comparer leurs résultats.

Le "Classification Learner" permet de tester les arbres de décision, l'analyse discriminante, les séparateurs à vaste marge, les K plus proches voisins et les méthodes ensemblistes, notamment.

Pour démarrer le "Classification Learner", il faut aller, dans la fenêtre principale de Matlab, dans l'onglet "APPS" et de le sélectionner dans la catégorie "Machine Learning and Deep Learning". Cela ouvre une grande fenêtre semblable à la Figure 10.11. Initialement, toutes les fonctionnalités sont inactives, il faut créer une nouvelle session et sélectionner les données à classifier depuis le workspace : le tableau de données doit contenir les données en lignes avec p colonnes pour les variables et une dernière colonne avec les étiquettes de catégorie. Après validation, Matlab revient à la fenêtre principale et affiche ces données de différentes couleurs en fonction des classes.

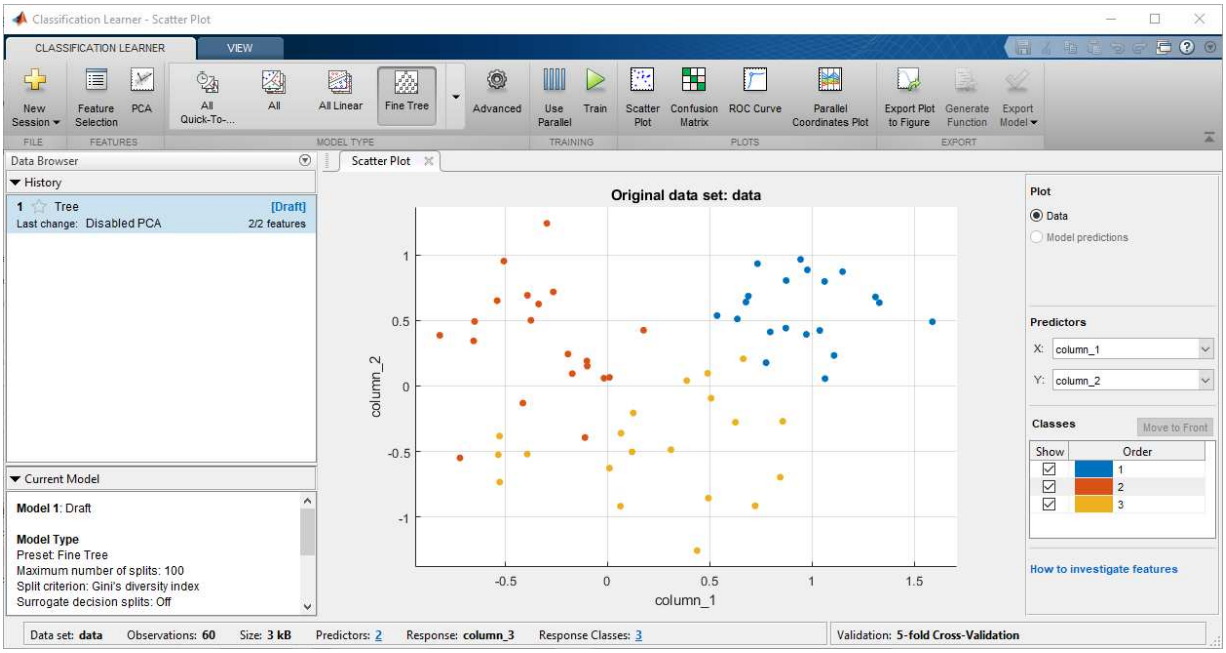


FIGURE 10.11 – Interface principale du Classification Learner, après avoir chargé un jeu de données.

Les différentes méthodes peuvent être sélectionnées dans le menu déroulant en haut à gauche de la fenêtre. Le bouton "Train" permet alors d'entrainer chacune de ces méthodes sur les données chargées. La liste des méthodes testées apparaît dans la liste "History" à gauche, avec pour chaque méthode le critère "Accuracy", qui est la proportion d'exemples correctement étiquetées par la méthode. Les réglages par défaut de chacune des méthodes extraites du menu déroulant peuvent être customisés en cliquant sur l'engrenage "Advanced" dans la barre supérieure des icônes.

Pour chaque méthode entraînée, 4 icônes permettent d'accéder à des visualisations des bonnes et mauvaises classifications :

- `Scatter Plot` (Figure 10.12 (a)) : cette figure présente les points en différentes couleurs en fonction de leurs classes d'origine. Les ronds correspondent aux classes correctement prédites et les croix correspondent aux erreurs de prédiction.
- `Confusion Matrix` (Figure 10.12 (b)) : La matrice de confusion présente, sur ses lignes, les classes réelles des données et, sur ses colonnes, les prédictions. Si le classificateur est parfait, cette matrice doit être purement diagonale et chaque case de la diagonale contient le nombre de données de chaque classe. On voit sur l'exemple de la Figure 10.12 (b) que 4 données ont été mal classées : 2 données de la classe 2 ont été classifiées dans la classe 3, alors que 2 données de la classe 3 ont été classées dans les classes 1 et 2 (voir section 6.1.1).
- `ROC Curve` (Figure 10.12 (c)) : la courbe ROC présente l'évolution du taux de vrais positifs en fonction du taux de faux positifs (voir section 6.1.3). Le classificateur est d'autant plus efficace que l'aire sous la courbe "AUC" (Area Under Curve) est proche de 1.
- `Parallel Coordinates Plot` (Figure 10.12 (d)) : sur cette figure, il y a autant de barres verticales que de dimensions. Pour cet exemple, les données sont dans $\mathbb{R}^2$, donc il y a deux barres verticales correspondant aux variables. Chaque donnée est représentée par une ligne reliant sa coordonnée x_1 normalisée sur la première barre à sa coordonnée x_2 sur la deuxième barre. Les lignes sont colorées en fonction de la classe de la donnée. ces diagrammes donnent une vue générale des domaines des classes dans l'espace des données. Les données mal classées y sont aussi représentées avec des symboles spécifiques.

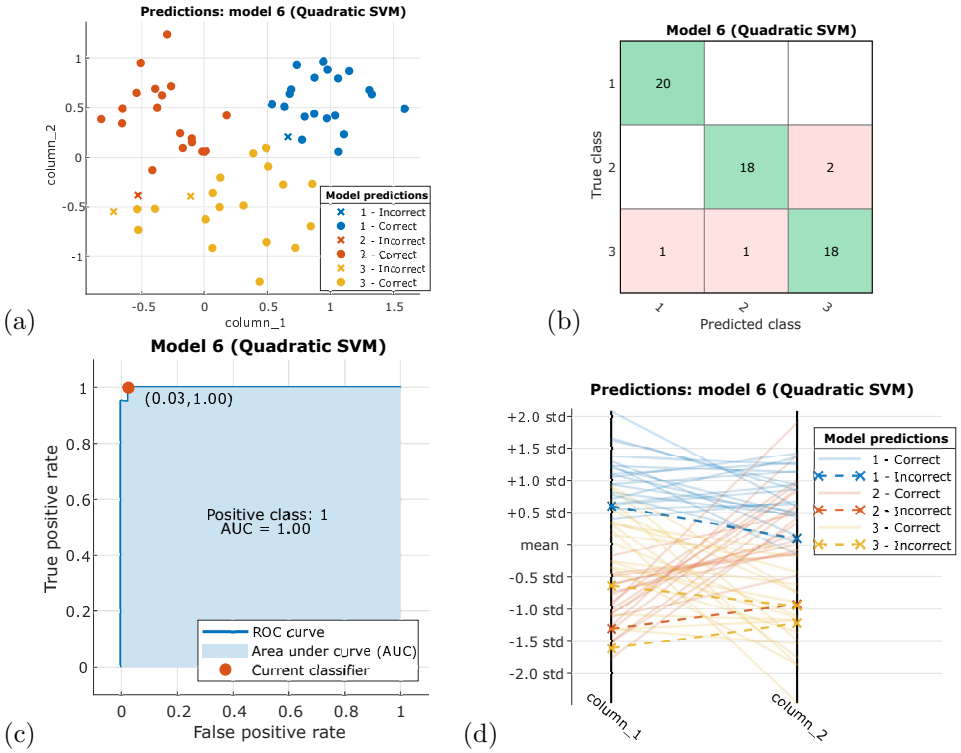


FIGURE 10.12 – Résultats de la classification de données.

Une fois que la méthode de classification a ainsi été choisie et éventuellement optimisée en termes d'hyperparamètres, il est possible d'exporter le classificateur entrainé sous forme d'un code pour lui soumettre d'autres données sans repasser par l'interface du "Classification Learner". Deux types d'exports sont possibles :

— Le bouton "Generate Function" crée une fonction, et l'ouvre dans les scripts Matlab, qui permettra de re-entraîner la méthode sélectionnée sur de nouvelles données, ou un ensemble plus large de données, directement en mode script. Cet export est conseillé pour pouvoir ensuite tout faire depuis un script.

— Le bouton "Export Model" exporte le classificateur entraîné dans le workspace où il pourra être utilisé pour faire de nouvelles prédictions. Par contre, il ne sera que dans le workspace et tout sera perdu à la fermeture de la fenêtre Matlab (ou en cas de bug quelconque). Le premier export sous forme de fonction est donc conseillé.

Cartes auto-adaptatives de Kohonen

Toute la création des cartes auto-adaptatives peut être réalisée par l'outil "Neural Networks" de Matlab : ces étapes d'entrianement de la carte sont détaillées en section 10.2.7. Après entrainement, la fenêtre d'évaluation permet de traiter cette carte pour "classer" de nouvelles données (Figure 10.13).

Dans la section en haut à droite "Optionnaly perform additional tests", il est possible de sélectionner des données depuis le workspace ou depuis un fichier. Le bouton "Test Network" exploite alors la carte entrainée précédemment pour placer les nouvelles données sur la carte. Les boutons "Plot SOM Neighbour Distances" et "Plot SOM Weight Planes" (Figures 10.7 (a) et (b)) rappellent uniquement les cartes de distances et de poids du réseau.

Les boutons "Plot SOM Sample Hits" et "Plot SOM Weight Positions" présentent les nouvelles données sur la carte auto-adaptative et dans l'espace de données (Figure 10.14 (a) et (b)).

Classification par réseau de neurones

La classification dans Matlab peut aussi se faire en mode script. Pour démarrer, il est utile de faire appel à l'outil "Neural Network" qui possède un mode "Pattern recognition and classification" : la commande `nnstart` dans la fenêtre de commande ouvre l'interface présentée sur la Figure 10.3, qui donne accès au mode Classification. Toutes les étaptes suivantes sont communes au mode Régression "Curve Fitting" et seront donc présentées de manière générale dans la section 10.2.10. La seule différence importante est liée à la structure du réseau. En mode "Fitting", le réseau par défaut utilise une couche de sortie à activation linéaire alors qu'en

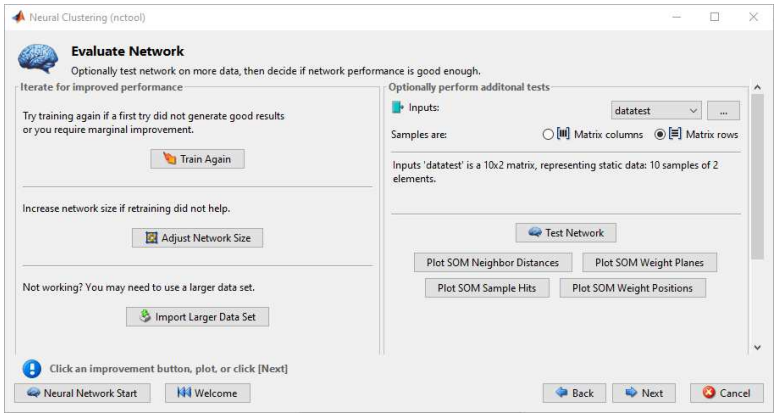


FIGURE 10.13 – Fenêtre d'exploitation des cartes auto-adaptatives de l'outil "Neural Networks" pour la classification avec carte de Kohonen.

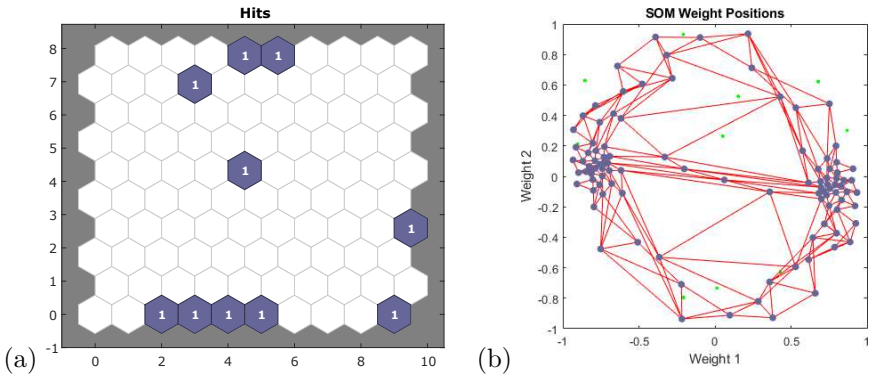


FIGURE 10.14 – Résultats de la classification de données sur une carte auto-adaptative.

mode "Classification", le réseau par défaut utilise une couche de sortie à activation `softmax` (voir section 6.5 et Figure 6.11). Cette différence nécessite de formater les données d'un certaine manière pour que l'outil "Neural Network" réussisse à comprendre les données.

Formatage pour la classification binaire

Considérons 8 points de $\mathbb{R}^2$ dans le carré $[0;1] \times [0;1]$ de coordonnées $(x_1; x_2)$ et appartenant soit à la classe A soit à la classe B. Connaissant les points, stockés en colonnes dans le tableau `DataInputs` (2×8), on construit le tableau `DataTargets` avec 2 lignes (car 2 classes) et 8 colonnes (car 8 données) avec des 0 et des 1 comme indiqué ci-dessous :

```
DataInputs = [0.6575 0.6799 0.2514 0.7332 0.5721 0.5864 0.4206 0.8572;  \% x_1
              0.7374 0.2379 0.8330 0.8245 0.6663 0.1079 0.4726 0.5604]; \% x_2
DataTargets = [ 1      1      0      1      1      1      0      1;  \%  classe A
                0      0      1      0      0      0      1      0]; \%  classe B
```

Dans cet exemple, la classe a été arbitraitement affectée telle que la classe est A si $x_1 > 0.5$ et B si $x_1 < 0.5$.

Formatage pour la classification multiclasse

Imaginons maintenant que ces mêmes données puissent appartenir à 4 classes, correspondant aux 4 quadrants du carré $[0;1] \times [0;1]$, alors on construira les tableaux de données tels que :

```
DataInputs = [0.6575 0.6799 0.2514 0.7332 0.5721 0.5864 0.4206 0.8572;  \% x_1
              0.7374 0.2379 0.8330 0.8245 0.6663 0.1079 0.4726 0.5604]; \% x_2
DataTargets = [ 0      1      0      0      0      1      0      0;  \%  classe A
                0      0      0      0      0      0      1      0;  \%  classe B
                0      0      1      0      0      0      0      0;  \%  classe C
                1      0      0      1      1      0      0      1]; \%  classe D
```

Une fois ces tableaux construits et stockés dans le workspace, il peuvent être chargés depuis l'outil "Neural Network" afin d'entrainer le réseau et d'extraire les résultats (voir section 10.2.10). Les données sont ici présentées en colonnes, mais elles peuvent aussi être formatées en lignes.

10.2.10 Réseaux de neurones : Perceptron monocouche

L'outil de référence pour s'initier aux réseaux de neurones dans Matlab est l'outil "Neural Network". Il peut être démarré depuis la fenêtre de commande en tapant `nnstart`. S'ouvre alors la fenêtre présentée sur la Figure 10.3. L'utilisation de cet outil, en mode "Régression", se fait en cliquant sur "Fitting App". Les variantes pour la classification ("Pattern Recognition app") et le clustering par cartes auto-adaptatives ("Clustering app") ont été présentées dans les sections 10.2.7 et 10.2.9.

Pour effectuer une régression par réseaux de neurones, les données doivent être formatées de manière adaptée à la structure du réseau. Matlab a besoin qu'on lui définisse les `Inputs` et les `Targets`. Pour une régression, un perceptron monocouche est suffisant : la structure minimale du réseau est de la forme $N_i - N_h - N_o$ où le nombre d'entrées N_i (les `Inputs`) et le nombre de sorties N_o (les `Targets`) sont définis par les données qu'on veut régresser. Si on veut régresser une conversion ($N_o = 1$) en fonction de la température, de la pression, de la fraction molaire d'un mélange binaire et de la concentration en catalyseur ($N_i = 4$), alors le tableau des entrées `Inputs` sera de taille $n \times 4$ et le tableau des sorties `Targets` sera de taille $n \times 1$, où n est le nombre de données. Les `Inputs` et `Targets` sont définis dans la fenêtre présentée sur la Figure 10.15 depuis le workspace.

Une fois que les données ont été déclarées, Matlab peut paramétrer le réseau et la méthode d'entrainement. La fenêtre "Validation and Test Data" (non présentée) permet de figer le pourcentage de données dédiées à l'entrainement, à la validation et au test. Par défaut, Matlab extraie 70 % pour l'entrainement, 15 % pour la validation et 15 % pour le test. Ces pourcentages sont une heuristique classique et peuvent être conservés pour démarrer.

Dans la fenêtre "Network Architecture" (non présentée), Matlab permet de choisir le nombre de neurones de la couche cachée, N_h. Par défaut, la valeur est de 10. Le nombre de neurones à choisir dépend du nombre de poids et biais à entrainer : le nombre de poids et biais peut être calculé pour une structure $N_i - N_h - N_o$ donnée (voir section 7.2.2). Il faut en général restreindre le nombre de neurones cachés pour que le nombre de poids et biais n'excède pas un dixième du nombre de données n (voir section 7.6.2). En première approximation, on peut choisir le nombre de neurones de la couche cachée tel que :

$$N_h \simeq \frac{n}{10(N_i + N_o)} \tag{10.1}$$

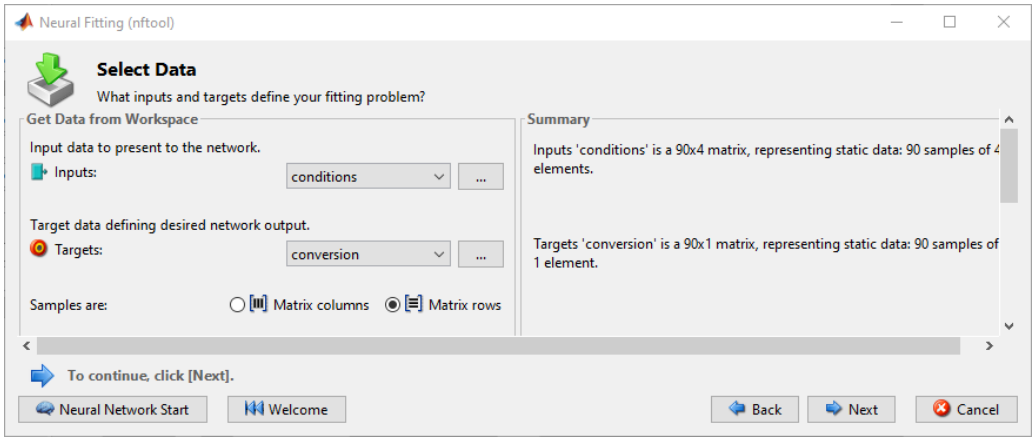


FIGURE 10.15 – Fenêtre de saisie des `Inputs` et `Targets` pour la régression par réseau de neurones.

Il est généralement contre-productif de mettre un grand nombre de neurones dans la couche cachée car cela emmène le réseau en sur-apprentissage : l'entrainement semblera très satisfaisant, mais le réseau sera incapable de généraliser sur de nouvelles données et il aura un caractère prédictif totalement aléatoire.

L'outil "Neural Networks" affiche alors la fenêtre d'entrainement "Train Network" (Figure 10.16). Elle permet de faire plusieurs entrainements successifs : comme l'initialisation change à chaque fois, le résultat peut varier d'un entrainement à un autre. Cette fenêtre permet de choisir l'algorithme d'entrainement (Levenberg-Marquardt par défaut, voir section 7.4.4). A droite apparaissent l'erreur quadratique moyenne MSE de chaque phase (entrainement, validation et test) et le coefficient de régression R (coefficient de détermination) entre les prédictions du réseau et les étiquettes vraies des données : une valeur proche de 1 de ce coefficient est recommandée pour obtenir une régression fidèle (voir section2.3.2).

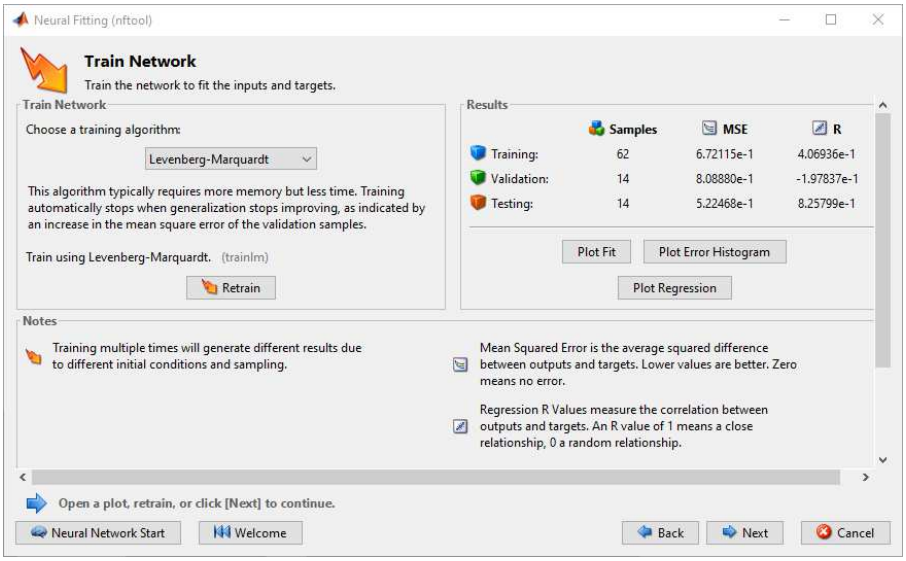


FIGURE 10.16 – Fenêtre d'entrainement d'un réseau de neurones.

A chaque fois que le bouton "Train" (ou "Retrain") est activé, il relance l'entrainement et modifie la fenêtre de la Figure 10.17 (a). Cette fenêtre fournit les détails relatifs à la phase d'entrainement et donne accès à plusieurs figures, notamment la courbe "Performance" représentée sur la Figure 10.17 (b). Cette figure présente l'évolution de l'erreur quadratique moyenne des lots de donées d'entrainement, de validation et de test au fur et à mesure des epochs. Ces courbes permettent de fixer les paramètres du réseau en choisissant l'époch qui fournit les meilleurs résultats. L'époch retenue est celle où l'erreur de validation commence à remonter. Cela signifie que le réseau entre en phase de sur-apprentissage : l'erreur d'entrainement est devenue très faible ce qui indique sur le réseau a capté les tendances principales des données et commence à apprendre à reproduire l'incertitude des données. L'erreur de validation remonte car le réseau n'arrive plus à généraliser sur des données nouvelles : sa prédiction est entâchée des incertitudes des données d'entrainement.

Les fenêtres suivantes permettent d'exporter le réseau de neurones entrainé vers le workspace ou sous forme

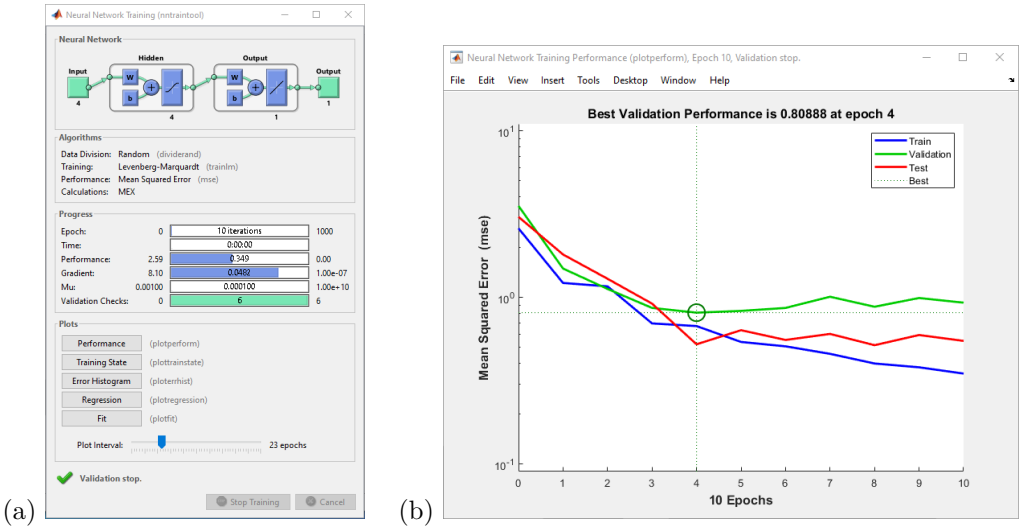


FIGURE 10.17 – Fenêtre de résultats d'un entrainement de réseau de neurones : (a) résumé des résultats d'entrainement, (b) détails de la figure "Performance".

d'un code pour être réentrainé ou pour prédire de nouvelles données sans repasser par l'interface.

La Figur 10.18 présente la forme générale d'un perceptron monocouche tel qu'il est construit dans Matlab. En mode script, les nombres de couches, les fonctions d'activation et les nombres de neurones par couche peuvent être fixés à loisir.

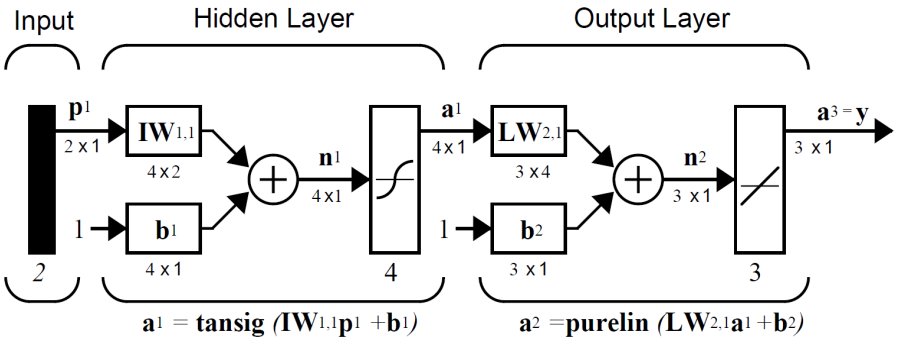


FIGURE 10.18 – Représentation d'un perceptron monocouhe dans Matlab.

10.2.11 Réseaux complexes et Deep Learning

Réseaux auto-encodeurs

Matlab ne possède par d'application dédiée à la construction et à l'entrainement de réseaux auto-encodeurs, mais il est possible de les définir très facilement en mode script.

La fonction `trainAutoencoder` permet de définir et entrainer un auto-encodeur à 1 seule couche cachée d'étranglement contenant `n_etranglement` neurones. La fonction doit recevoir un tableau `data` où les données sont stockées en colonnes : n données dans $\mathbb{R}^p$ seront donc stockées dans un tableau `data` de taille $p \times n$. Le code minimal s'écrit :

```
data = load("mes_donnees.txt")
n_etranglement = 10;
mon_reseau = trainAutoencoder(data, n_etranglement);
data_predites = predict(mon_reseau,data);
mseError = mse(data - data_predites)
view(mon_reseau)
```

— La fonction `trainAutoencoder` permet de créer et entrainer le réseau autoencodeur. Le résultat est stocké dans une structure `mon_reseau` qui contient, par exemple, les poids et biais d'encodage, `mon_reseau.EncoderWeights` et `mon_reseau.EncoderBiases`, les poids et biais de décodage, `mon_reseau.DecoderWeights` et `mon_reseau.DecoderBiases`. L'appel à la fonction `trainAutoencoder` fait apparaître la fenêtre d'entrainemenent du réseau qui permet de suivre son évolution et d'accéder aux graphes de performance.
— La fonction `predict` crée le tableau des prédictions, de même taille que `data`.
— La fonction `mse` calcule l'écart quadratique moyen entre les données originales de data et les prédictions `data_predites`.
— La fonction `view` permet de visualiser le réseau.

Si un réseau auto-encodeur à une seule couche cachée d'étranglement, $N_i - N_e - N_i$, n'est pas adapté à la réduction de dimensionnalité, il est possible d'inclure des couches supplémentaires pour avoir une structure $N_i - N_{h1} - N_e - N_{h1} - N_i$ (voir section 5.3). L'entrainement doit alors se faire itérativement comme indiqué sur la Figure 5.7 : on entraine d'abord un premier réseau $N_i - N_{h1} - N_i$, puis on entraine ensuite $N_{h1} - N_e - N_{h1}$. Les principales étapes du code sont alors :

```
mon_reseau1 = trainAutoencoder(data,Nh1);
latentes1 = encode(mon_reseau1,data);
mon_reseau2 = trainAutoencoder(latentes1,Ne);
latentes2 = encode(mon_reseau2,latentes1);
```

— Le premier appel à `trainAutoencoder` permet de créer et entrainer le réseau $N_i - N_{h1} - N_i$,
— Le premier appel à `encode` permet d'encoder les données originales `data` selon la partie "Encodeur" pour accéder aux variables latentes de sa couche d'étranglement.
— Ces variables latentes du premier réseau sont utilisées comme données d'entrée pour l'entrainement du réseau $N_{h1} - N_e - N_{h1}$ grâce au second appel à `trainAutoencoder`.
— Le second appel à `encode` permet de connaître les variables latentes dans l'espace réduit à N_e dimensions.

Deep Learning App

Matlab possède une application Deep Learning interne, qui permet de créer des réseaux complexes en incluant des couches extraites d'une bibliothèque assez large. N2anmoins son fonctionnement est complexe et sort largement du cadre de ce cours. Cette application ne sera donc pas présentée ici.

Exercices

11.1 Préparation de données

11.1.1 Analyse d'une matrice de projections

On considère les 100 données non étiquetées en 4 dimensions $(x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, x_4^{(i)})$ présentées sur la Figure 11.1 sous forme de leur matrice de projections.

1. Quelles informations fournit cette matrice sur d'éventuelles corrélations entre les variables ?
2. Que pourrait-t-on proposer pour réduire la dimensionnalité des données ?

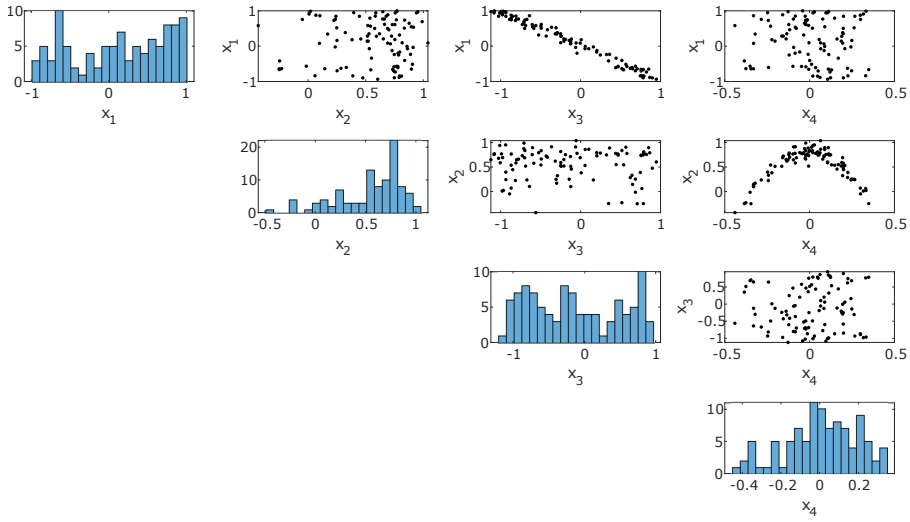


FIGURE 11.1 – Matrice de projections de données.

11.1.2 Matrices de covariance et de corrélation

On considère les données présentées sur la Figure 11.1. La matrice de covariance de ces données, $\boldsymbol{\Sigma}$, et leur matrice de corrélation, $\boldsymbol{R}$, sont respectivement :

$$\boldsymbol{\Sigma} = \begin{pmatrix} 0.2590 & -0.0043 & -0.2576 & 0.0071 \\ -0.0043 & 0.1292 & 0.0028 & -0.0189 \\ -0.2576 & 0.0028 & 0.2649 & -0.0074 \\ 0.0071 & -0.0189 & -0.0074 & 0.0426 \end{pmatrix} \tag{11.1}$$

$$\boldsymbol{R} = \begin{pmatrix} 1.0000 & -0.0235 & -0.9832 & 0.0680 \\ -0.0235 & 1.0000 & 0.0150 & -0.2547 \\ -0.9832 & 0.0150 & 1.0000 & -0.0694 \\ 0.0680 & -0.2547 & -0.0694 & 1.0000 \end{pmatrix} \tag{11.2}$$

Discuter les valeurs de ces matrices en relation avec la matrice de projection de la Figure 11.1. L'une de ces représentations (matrice de projection, matrice de covariance, matrice de corrélation) semble-t-elle plus informative que les autres ?

11.1.3 Normalisation de données brutes

On considère les données brutes présentées sur la Figure 11.2. Quelles méthodes de normalisation peut-on proposer pour ces variables ? La même méthode de normalisation peut-elle être appliquée à chaque variable ou est-il préférable de leur appliquer des méthodes différentes ?

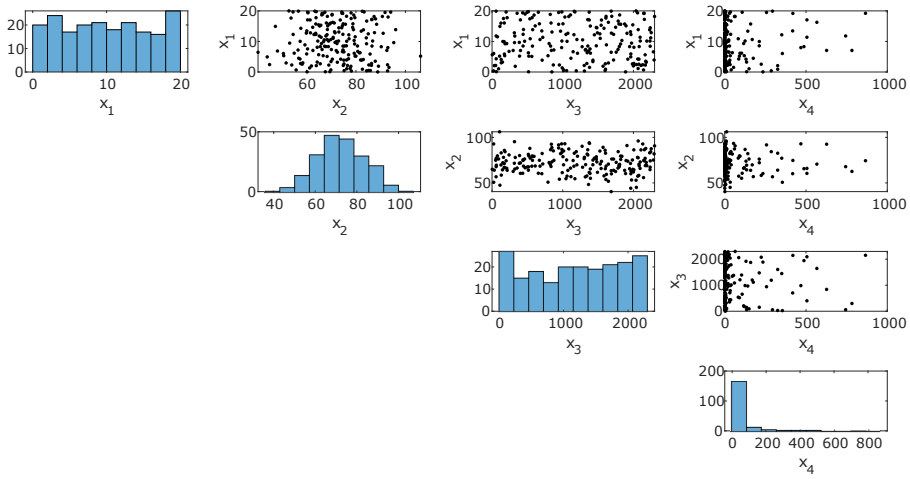


FIGURE 11.2 – Matrice de projections de données à normaliser.

x_1	x_2	x_3	x_4
15.09	68.59	610.5	0.1206
18.68	66.19	1109.8	0.0268
10.62	68.40	679.0	1.5557
10.15	72.96	2096.4	24.102
17.99	92.99	1030.2	0.0257
1.34	61.29	993.0	0.5379
19.58	94.04	1374.4	3.1775
⋮	⋮	⋮	⋮
2.26	69.03	2168.6	0.0193
3.40	69.93	1426.4	0.4115
8.31	68.85	2081.7	12.493
2.08	63.99	30.8	325.51
7.35	70.07	1324.1	0.0130
0.44	75.21	1054.9	0.0158
3.93	76.19	1719.1	1.8197
10.51	67.20	1281.0	24.198

TABLE 11.5 – Extrait du tableau de données.

11.1.4 Régression de données temporelles

On considère les 100 données temporelles présentées sur la Figure 11.3. On souhaite régresser l'étiquette y en fonction des variables x_1 à x_5 par une régression multilinéaire. Quelles sont les caractéristiques problématiques de ces données ? Combien de données peut-on conserver pour la régression ?

11.1.5 Evolution des parcs régionaux éolien et solaire (2001-2019)

La Figure 11.4 présente les évolutions des puissances disponibles des parcs éolien et solaire des 13 régions françaises (2001-2019). Ces courbes ont été construites à partir de données publiques disponibles sur `https://opendata.reseaux-energies.fr`. L'objectif de cet exercice est de tracer cette figure à partir d'un fichier simplifié de ces données.

Les étapes successives à suivre pour construire cette figure particulière sont :

— Télécharger le fichier `parc-regional-annuel-prod-eolien-solaire_mini2.csv` sur Arche,

— Créer un script Matlab et lire le fichier avec `readtable` et le stocker dans une variable appelée `data`,

— Explorez les propriétés de cette table avec `data.Properties`,

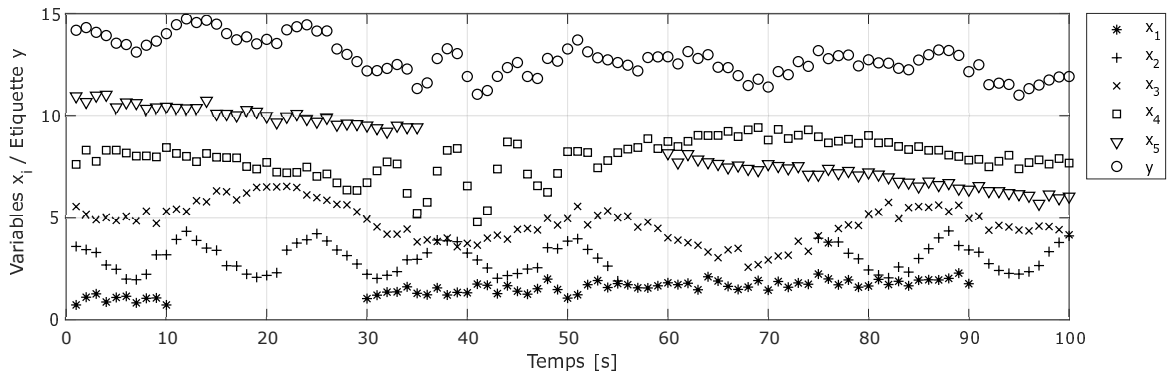


FIGURE 11.3 – Données temporelles à régresser.

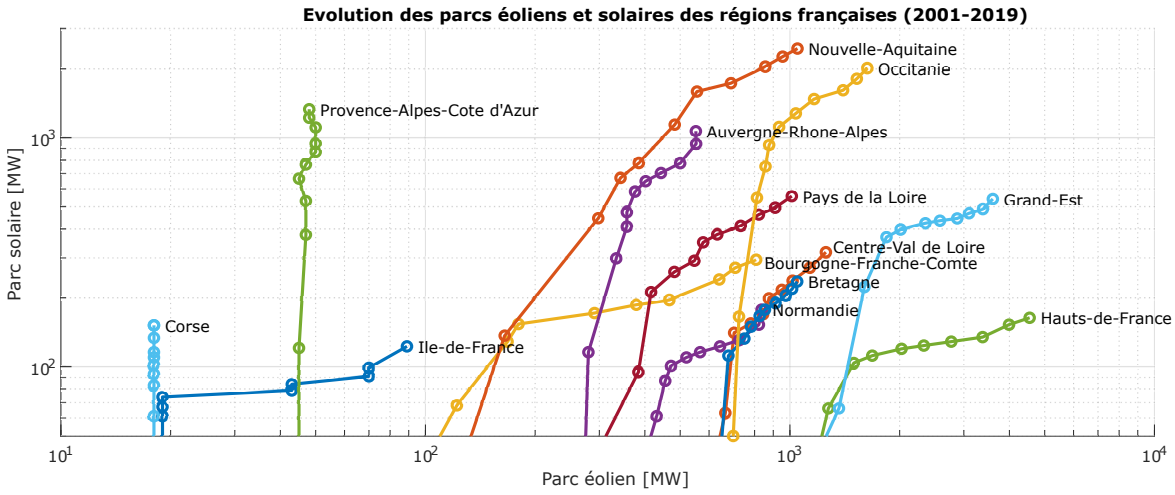


FIGURE 11.4 – Evolution des puissances disponibles des parcs éolien et solaire des 13 régions françaises (2001-2019).

— Extraire les 4 colonnes numériques, contenant l'année, le numéro INSEE de région, la puissance éolienne et la puissance solaire, vers un tableau `"donnees"` avec `table2array`,
— Trouver la liste de tous les numéros de régions, et leurs indices, grâce à la fonction `unique`,
— Extraire de la colonne "Région" de la table `data` les noms de ces régions, et les stocker dans un tableau de chaîne de caractères,
— Créer une nouvelle figure,
— Pour chaque région i :
  — Trouver dans `donnees` toutes les lignes correspondant à la région i grâce à la fonction `find`,
  — Extraire toutes ces lignes vers un tableau `prodregion` en ne gardant que les colonnes donnant l'année, la puissance éolienne et la puissance solaire,
  — Trier les lignes de ce tableau par ordre d'année croissante avec `sortrows`,
  — Tracer la colonne de `prodregion` donnant la puissance solaire en fonction de celle donnant la puissance éolienne, en la superposant aux éventuelles courbes déjà tracées,
  — Ajouter à côté de la dernière donnée une étiquette texte affichant le nom de cette région, avec la fonction `text`.
— Finaliser la figure avec les titres des axes, le titre général, les échelles, etc.

Une fois la figure tracée, on peut chercher à la comprendre :
— Quelle est la région qui possède le plus grand parc renouvelable, éolien et solaire confondus ?
— Pourquoi les régions PACA et Hauts-de-France ont-elles des évolutions diamétralement opposées ?
— Pourquoi les trajectoires de la Corse et PACA sont-elles verticales ?
— Pourquoi la région Ile-de-France semble-t-elle à la traine par rapport aux autres ?
— Pourquoi la Corse produit-elle plus d'énergie solaire que l'Ile-de-France, alors que l'Ile-de-France est 50 % plus étendue et 35 fois plus peuplée ?

11.2 Clustering, partitionnement

11.2.1 Clustering par K-moyennes de données régulières

On considère les données représentées par des carrés sur la Figure 11.5. On souhaite leur appliquer la méthode des K-moyennes pour identifier 2 clusters. Les deux figures représentent deux propositions d'initialisation des centroïdes, symbolisés par les carré noirs. Quels clusters seront détectés par ces deux initialisations ? La méthode pourrait-elle converger vers d'autres partitions des données ? Si oui, pour quelles initialisations ?

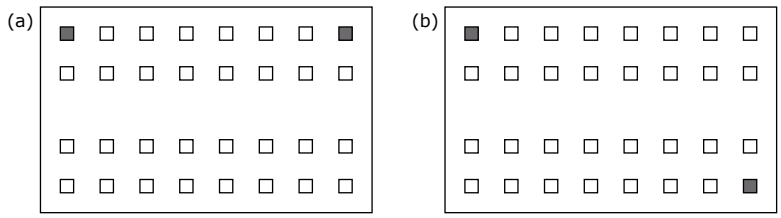


FIGURE 11.5 – Données à classer en deux clusters par la méthode des K-moyennes.

11.2.2 Choix d'une méthode de clustering

Quelle méthode de clustering peut-on conseiller/déconseiller pour partitionner les données présentées sur chaque cas de la Figure 11.6 ? On pourra évaluer la pertinence de la méthode des K-moyenne, le clustering hiérarchique, la méthode DBSCAN, les modèles de mélange gaussiens et le mean-shift clustering.

11.2.3 Histogrammes de distances entre données

La Figure 11.7 présente les histogrammes des distances deux-à-deux des données présentées sur la Figure 11.6. Discuter de la pertinence de ces histogrammes pour extraire une information sur la forme et les propriétés de clusters présents parmi ces données.

11.2.4 Procédé de carbonatation

On s'intéresse au procédé de carbonatation présenté sur la Figure 11.8. La ligne de production est constituée de deux cuves agitées continues à double enveloppe, qui agissent en série. La première cuve est appelée le

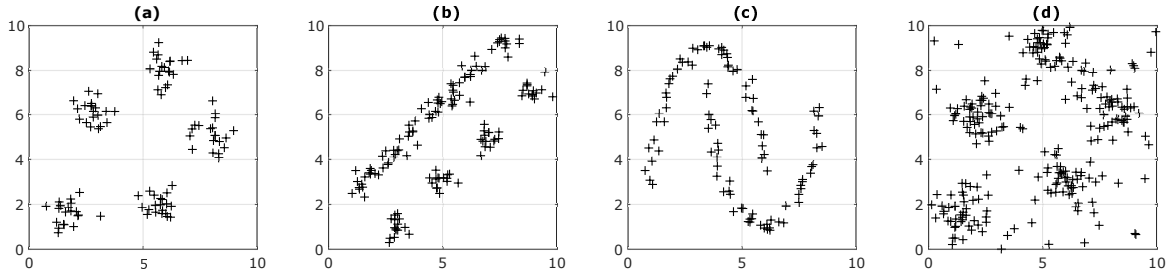


FIGURE 11.6 – Données à partitionner.

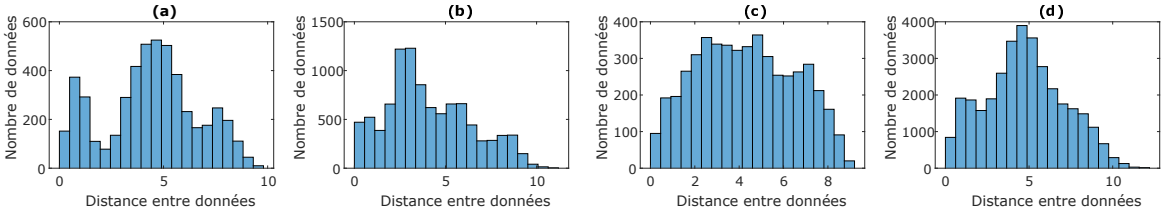


FIGURE 11.7 – Histogrammes de distances deux-à-deux des données à partitionner.

"carbonateur" et la seconde le "finisseur". Chaque cuve est alimentée avec du CO_2 et refroidie par un circuit d'eau froide.

Un système de commande permet d'ajuster le débit d'eau de refroidissement Q_{eau}, et les vannes de distributions de l'eau, α, et du CO_2, β. Les deux concentrations $C_{A,1}$ et $C_{A,2}$, ainsi que les 4 températures T_1, T_2, $T_{cal,1}$ et $T_{cal,2}$ sont mesurées et leurs valeurs historiques sont stockées sur un serveur central. Les valeurs historiques de Q_{eau}, α et β ne sont pas disponibles.

A plusieurs reprises, le procédé a connu des dérives significatives, mais temporaires : la situation est revenue à la normale sans que l'origine ait encore pu être identifiée. L'une de ces dérives a failli nécessiter l'arrêt d'urgence du procédé. On cherche ici à identifier ces dérives et éventuellement leurs origines : les opérateurs soupçonnent la pompe de refroidissement de faiblir et les vannes de distribution de se bloquer partiellement.

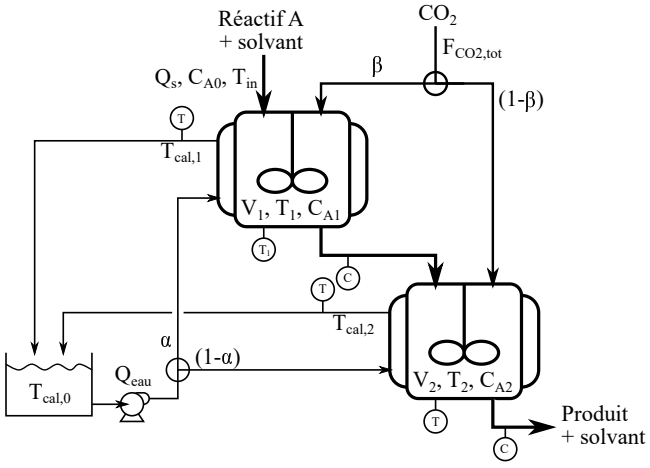


FIGURE 11.8 – Schéma du procédé de carbonatation.

1. Récupérer sur Arche le fichier `donnees_TC_carbonatation.txt` et charger son contenu vers une variable `data` dans un code Matlab,
2. Faire un tracé rapide des différentes colonnes de ce tableau pour visualiser leurs plages de variations,
3. Normaliser chacune des variables sur la plage $[-1; 1]$,
4. Calculer le vecteur des distances avec `pdist`. Attention, ce vecteur peut être de très grande taille : il peut être préférable de tester d'abord le code sur une partie des données avant de l'appliquer à la totalité. Tracer ensuite l'histogramme de la distribution des distances entre paires de données : peut-on s'attendre à la présence de clusters ?

5. Récupérer la fonction `fonction_matrice_projections.m` sur Arche et l'utiliser pour tracer la matrice de projections et distribution des données `data`. Que nous apprend cette matrice sur l'existence et les propriétés des clusters ? Choisir dès à présent deux projecitons qui semblent présenter le plus clairement les éventuels clusters.
6. Utiliser le fonction `kmeans` pour tester une premier clustering par la méthode des K-moyennes avec un nombre de clusters K au choix.
7. Dans une boucle pour un nombre de clusters en 2 et 8, faire des appels répétitifs à kmeans de façon à tester l'impact de l'initialisation automatique. Pour chaque réponse de `kmeans`, ajouter un point sur une figure présentant la somme des distances intra-clusters en fonction du nombre de clusters K. L'initialisation a-t-elle un impact sur la réponse de la méthode ? Combien de clusters, K_{opt}, doit-on choisir ?
8. Refaire appel à `kmeans` avec le nombre définitif de clusters K_{opt} : pour éviter le biais d'initialisation, faire une vingtaine d'appels et sélectionner les réglages de celui qui fournit la distance intra-clusters minimale.
9. Tracer une figure avec les deux projections retenues précédemment pour bien voir les éventuels clusters. Projeter toutes les données sur ces coupes en modifiant la couleur des points en fonction du cluster. Superposer les centroïdes. Le nombre de clusters K_{opt} vous parait-il être le meilleur : la méthode des K-moyennes serait-elle passe à côté d'une partie du problème ?

On souhaite ensuite tester la méthode de clustering hiérarchique, pour tracer le dendrogramme des données et vérifier si le choix proposé par les K-moyennes était effectivement judicieux :

1. Faire appel aux fonctions `pdist`, `linkage` et `dendrogram` pour tracer le dendrogramme des données. Le choix du nombre de clusters par la méthode des K-moyennes est-il confirmé par ce dendrogramme ?

On souhaite enfin tester la méthode DBSCAN :
— Faire appel à la fonction `dbscan` après avoir fixé rationnellement les valeurs des paramètres `epsilon` et `minpts` pour le rayon de voisinage et le nombre minimal de voisins. Projeter ensuite les clusters ainsi identifiés sur les mêmes plans de coupe que précédemment.
— Faire varier manuellement les valeurs de `epsilon` et `minpts` pour voir leur impact sur le nombre de clusters identifiés et le nombre de points considérés comme isolés.

11.2.5 Etude des hyper-paramètres d'une méthode DBSCAN

On souhaite affiner les réglages des hyper-paramètres d'une méthode DBSCAN, à savoir `epsilon` et `minpts` qui définissent le rayon de voisinage et le nombre minimal de voisins (voir section 4.5). On utilisera des données partielles issues du procédé de carbonatation (voir Exercice 11.2.4).

1. Charger le fichier `donnees_check_DBSCAN.txt` depuis Arche et charger son contenu dans un tableau `data` depuis un script Matlab. Stocker le nombre total de données dans une variable `ndata`.
2. Faire un tracé rapide des données brutes. Comme ce sont des données dans $\mathbb{R}^3$, projeter les données sur 3 plans $(x_1; x_2)$, $(x_1; x_3)$ et $(x_3; x_2)$ placés de façon appropriée.
3. Faire une première application de `dbscan` en fixant `epsilon = 0.1` et `minpts = 5`. Retracer les donnnées sur ces mêmes plans de projection, en utilisant la fonction `gscatter` pour colorer les points en fonction de leur numéro de cluster d'affectation.
4. Stocker dans une variable `frac_pts_isoles` la fraction de points isolés en comptant les -1 dans le vecteur d'incices renvoyé par `dbscan`, et dans `nb_clusters` le nombre de clusters identifiés en cherchant l'indice maximal dans ce vecteur.
5. On souhaite ensuite faire varier `epsilon` et `minpts` pour voir leurs impacts sur le nombre de clusters et sur la fraction de points isolés. Pour cela, créer deux vecteurs contenant les différentes valeurs qu'on souhaite balayer pour chacune de ces grandeurs,
6. Puis, construire une double boucle permettant de balayer tous les couples de valeurs des deux paramètres. A chaque tour de cette double boucle :
   — affecter aux variables `epsilon` et `minpts` leurs valeurs temporaires correspondant au tour en cours,
   — appeler `dbscan` avec ces valeurs et stocker le vecteur d'indices de clusters renvoyé,
   — analyser ces indices de clusters (comme précédemment) pour déterminer les valeurs de `frac_pts_isoles` et `nb_clusters` correspondant aux valeurs temporaires de `epsilon` et `minpts` de ce tour de boucle.
   — Stocker ces valeurs dans deux tableaux à des positions correspondant au tour de boucle en cours.
7. Un fois que toutes les valeurs de `epsilon` et `minpts` ont été testées, tracer deux cartes, grâce aux fonctions `contour` et `contourf` :

— la carte présentant les courbes iso-valeurs de la fraction de points isolés en fonction de `epsilon` et `minpts`,
— la carte présentant les courbes iso-valeurs du nombre de clusters identifiés en fonction de `epsilon` et `minpts`. Comme le nombre de clusters est un entier, il est préférable de tracer des isovaleurs non-entières sur cette carte pour mieux délimiter les domaines de prédominance des entiers (Figure 11.9).

8. Discuter ces deux cartes en répondant aux questions suivantes :
— Vers quelle valeur tend le nombre de clusters identifiés lorsque `epsilon` devient grand ?
— Vers quelle valeur tend le nombre de clusters identifiés lorsque `minpts` devient grand ?
— Pourquoi le nombre de clusters identifiés devient-il très grand lorsque `epsilon` et `minpts` deviennent petits ? Quelle est la valeur limite maximale de ce nombre de clusters ?
— Pourquoi la fraction de points isolés devient-elle grande lorsque `epsilon` est petit et `minpts` grand ?
— Quel choix final de `epsilon` et `minpts` peut-on recommander pour rejeter moins de 10 % des données (points isolés) tout en détectant un nombre de clusters cohérent avec la visualisation des données ?

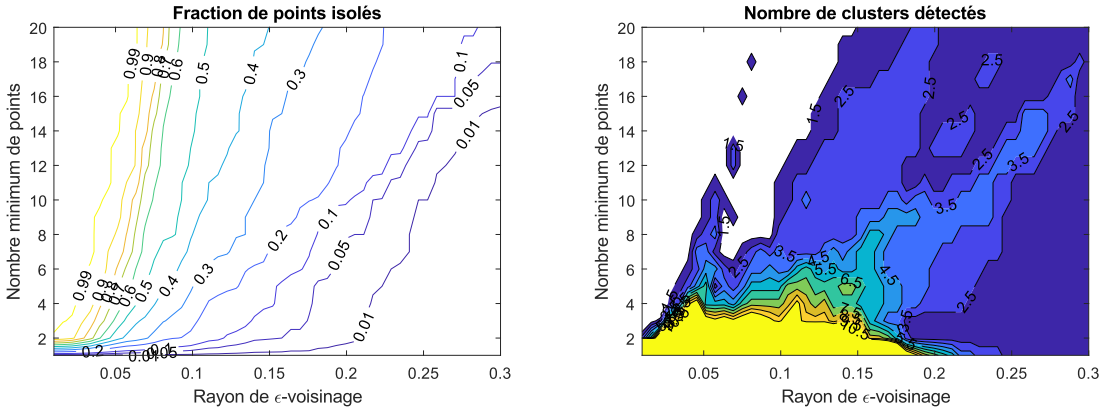


FIGURE 11.9 – Cartes obtenues en balayant les hyperparamètres de DBSCAN.

11.2.6 Profils types d'élèves-ingénieurs

On souhaite analyser les résultats académiques d'une promotion d'élèves-ingénieurs ENSIC pour vérifier s'il existe, ou non, des profils types en termes de modules préférés : y a-t-il des élèves qui aiment la chimie organique et pas l'informatique et d'autres qui aiment le management et pas la thermodynamique ? Si de tels profils types existent, on devrait pouvoir les détecter soit sous formes de clusters, soit via des corrélations entre les notes obtenues dans les modules correspondants.

1. Télécharger sur Arche le fichier `notes_promo_XXXX_S5S6.txt` et stocker son contenu dans un tableau temporaire `datatemp`. La première colonne contient le numéro de l'étudiant et les colonnes suivantes contiennent les notes dans les 12 modules du semestre 5 et du semestres 6.
2. Stocker le nombre d'élèves dans une variable `ndata`.
3. Extraire de `datatemp` uniquement la partie correspondant aux notes et stocker ces notes dans `data`.
4. Pour se focaliser sur les "goûts" personnels des élèves sans que l'analyse soit influencée par leurs niveaux individuels, on doit d'abord normaliser les profils de notes de chaque élève. Pour chaque ligne du tableau, calculer la moyenne `moyeleve` et l'écart-type `stdeleve` des notes de l'élève dans tous les modules. Normaliser toute la ligne d'un élève pour en faire des notes centrées réduites. Une note positive signifie que l'élève surperforme dans ce module par rapport à son niveau global : ce module est son point fort.
5. Tracer rapidement les différentes colonnes de data avec la fonction `plot` : voit-on des modules sortir du lot ? Y a-t-il des modules où tout le monde surperforme ou sous-performe ?
6. Calculer le vecteur `distrib` des distances entre données deux-à-deux, et tracer l'histogramme de ces distances : cet histogramme nous renseigne-t-il sur l'existence de clusters ?
7. Appliquer la méthode DBSCAN aux données pour chercher des clusters : choisir les paramètres `epsilon` et `minpts` de manière rationnelle et faire différents essais pour voir si certains réglages permettent de trouver des clusters.
8. Calculer avec `corrcoef` la matrice des coefficients de corrélations entre notes de modules. Remplacer sa diagonale de 1 par des 0. Tracer l'image de cette matrice avec `image`. Y a-t-il des corrélations entre

modules ? Peut-on dire "Les élèves qui sont bons dans le module XXX sont aussi bons dans le module YYY" ou "les élèves qui sont bons en XXX sont mauvais en YYY" ?

11.3 Réduction de dimensionnalité

11.3.1 Nombres de composantes principales d'une ACP

On dispose de 100 données dans un espace à 7 dimensions. On souhaite en faire une ACP pour réduire la dimension, mais en conservant suffisamment de composantes principales pour parvenir à expliquer 90 % de la variance totale des données :

1. Combien de composantes principales devra-t-on conserver si toutes les données sont presque parfaitement alignées selon une droite ?
2. Combien de composantes principales devra-t-on conserver si toutes les données sont presque parfaitement coplanaires ?
3. Combien de composantes principales devra-t-on conserver si les données sont complètement aléatoires ?

11.3.2 Récupération de données après réduction de dimensionnalité

Une analyse en composantes principales a été appliquée à des données dans un espace à 9 dimensions. L'ACP a permis de retenir 3 composantes principales pour expliquer 90 % de la variance globale. Pour réduire l'espace de stockage, on souhaite ne conserver que les composantes principales des données. Pourra-t-on ensuite reconstruire les données originales ?

11.3.3 Réduction de dimensionnalité ?

On dispose de n données, D $\{x^{(i)}, y^{(i)}\}_{i=1,\dots,n}$ où $x^{(i)}$ possède 6 dimensions $x_1^{(i)}$, $x_2^{(i)}$, $x_3^{(i)}$, $x_4^{(i)}$, $x_5^{(i)}$ et $x_6^{(i)}$ pour chaque donnée i. On est parvenu à entrainer un perceptron monocouche avec les entrées $\{x_1^{(i)} + x_2^{(i)}; x_2^{(i)} + x_3^{(i)} - x_4^{(i)}; x_6^{(i)} - x_5^{(i)}\}_{i=1,\dots,n}$, qui reproduit fidèlement les étiquettes $y^{(i)}$.

1. S'agit-il d'une réduction de dimensionnalité ?
2. Si on avait appliqué une ACP avant d'entrainer le réseau de neurones, combien de composantes auraient permis d'expliquer l'essentiel de la variance totale ?

11.3.4 Taille des couches cachées d'un réseau auto-encodeur multicouche

On considère le réseau auto-encodeur symétrique à couches cachées multiples représenté sur la Figure 11.10. Les données sont situées dans un espace à 100 dimensions et on souhaite tester la possibilité de les compresser dans deux variables latentes.

1. Combien de neurones faut-il placer dans la couche cachée de l'encodeur (et du décodeur) si on utilise une seule couche cachée ? Quel est alors le nombre de paramètres du réseau (poids et biais) ?
2. Combien de neurones faut-il placer dans chacune des couches cachées si on utilise 2 couches cachées ? Quel est alors le nombre de paramètres du réseau ?
3. Combien de neurones faut-il placer dans les couches cachées si on utilise 3 couches ? Quel est alors le nombre de paramètres du réseau ?

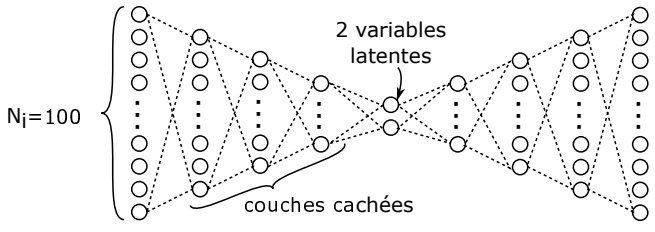


FIGURE 11.10 – Représentation d'un réseau auto-encodeur à couches cachées multiples.

11.3.5 ACP sur des données dans $\mathbb{R}^6$

1. Charger depuis ARCHE le fichier `data_ACP_6D.txt` et stocker son contenu dans un tableau `data`.
2. Faire un tracé rapide des projections des données avec la fonction `gplotmatrix`, et calculer la matrice des coefficients de corrélation entre ses colonnes avec la fonction `corrcoef`. Les variables qui constituent les données sont-elles indépendantes ou existe-t-il des corrélations entre elles ? A combien de dimensions peut-on raisonnablement espérer réduire la taille de l'espace des variables ?
3. Réaliser une analyse en composantes principales de ces données grâce à la fonction `pca` afin d'obtenir les vecteurs des composantes principales, les coordonnées des projections des données sur cette nouvelle base de vecteurs dans un tableau `projections`, les variances associées à chaque composante, ainsi que les fractions de variance dans un vecteur `fracvariances`.
4. Tracer avec `gplotmatrix` les nouvelles données dans la base des composantes principales et avec `bar` l'histogramme des fractions de variance expliquées par chaque composante principale. A combien de dimension peut-on réduire l'espace des variables pour expliquer près de 99 % de la variance originale des données ? Expliciter les nouvelles variables en fonction des anciennes sous la forme :

$$CP_j = a_{1,j}x_1 + a_{2,j}x_2 + a_{3,j}x_3 + a_{4,j}x_4 + a_{5,j}x_5 + a_{6,j}x_6 \tag{11.3}$$

11.3.6 Carte de Kohonen de données dans $\mathbb{R}^3$

1. Charger depuis ARCHE le fichier `data_carte_kohonen.txt` et stocker son contenu dans un tableau `data`.
2. Faire un tracé rapide des projections des données avec la fonction `gplotmatrix`.
3. Démarrer l'outil "Neural Networks" avec la fonction `nnstart` dans la fenêtre de commande. Sélectionner le mode "Clustering". Sélectionner le tableau `data` dans le workspace et préciser que les données sont en ligne. Commencer avec une carte de taille 10.
4. Faire différents essais d'entrainement en changeant manuellement la taille de la carte. Observer les résultats sur les cartes de distance entre voisins et sur le tracé de la position des poids dans l'espace des variables.
5. Générer automatiquement un script simple contenant les principales fonctions matlab pour construire et entrainer une carte auto-adaptative.

11.4 Classification

11.4.1 Classification à 2 classes par réseaux de neurones

On considère les données des Figures 11.11 (a) à (d), que l'on souhaite classer grâce à des perceptrons monocouches, de type $2 - N_H - 1$. Les deux entrées sont les coordonnées x_1 et x_2 et la sortie est l'étiquette représentant la classe : 0 pour les + et 1 pour les ∘. Les N_H neurones de la couche cachée possèdent une activation sigmoïde et le neurone de sortie une activation linéaire. Pour chaque exemple, estimer le nombre de neurones nécessaires pour classer correctement les données selon leur classe. Lorsque c'est possible, estimer les valeurs des poids synaptiques et des biais.

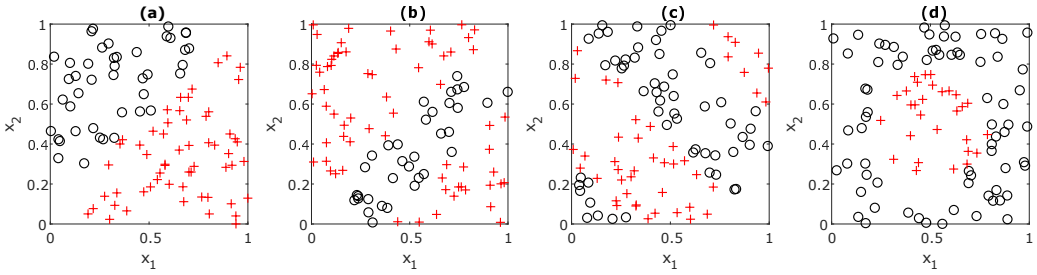


FIGURE 11.11 – Exemples de données à classer par réseau de neurones.

11.4.2 Courbe ROC d'une classification binaire

On considère les données de classification binaire présentées dans le Tableau 11.6. En utilisant une simple classification par seuil ($x < x_{seuil} \Rightarrow y = 0$; $x > x_{seuil} \Rightarrow y = 1$), construire la courbe ROC de ce classificateur.

Variable x	5	8	12	13	15	19	21	27	28	30
Etiquette	0	0	1	0	1	0	1	0	1	1

TABLE 11.6 – Données pour la construction d'une courbe ROC.

11.4.3 Détection automatique de régimes hydrodynamiques en microcanal

On s'intéresse à l'hydrodynamique d'un écoulement gaz-liquide dans un microcanal de diamètre D = 180 μm. Le suivi de l'écoulement par une caméra rapide et l'analyse d'images ont permis de détecter 4 régimes hydrodynamiques (Figure 11.12, gauche) et de tracer une carte de ces régimes en fonction des vitesses superficielles des phases gaz et liquide (Figure 11.12, droite). L'objectif de cet exercice est de d'établir une classification de ces régimes afin de prédire automatiquement le régime en fonction des vitesses superficielles sur de nouvelles données.

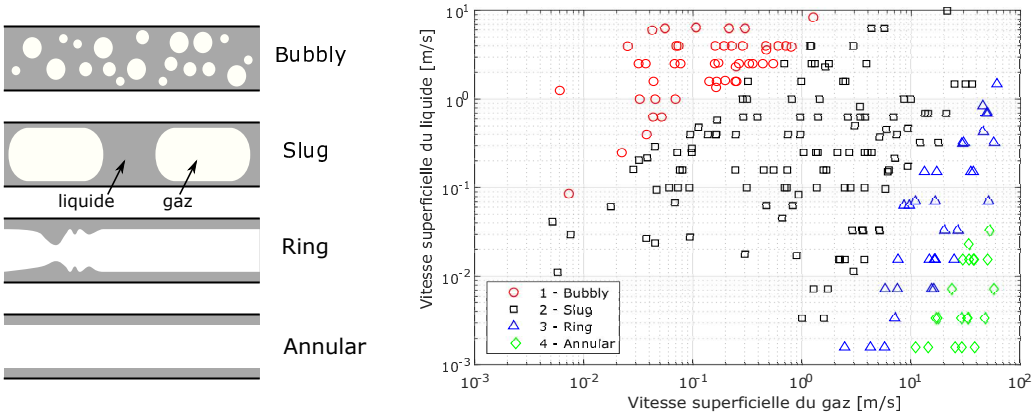


FIGURE 11.12 – Représentation de 4 régimes hydrodynamiques gaz/liquide en microcanal (gauche) et cartes de régimes en fonction des vitesses superficielles des fluides (droite).

1. Télécharger les deux fichiers de données : "`data_regimes_hydrodynamiques.txt`" qui contient les données étiquetées de la carte pour entrainer le classificateur, et "`data_regimes_a_predire.txt`" qui contient les nouvelles données dont on veut prédire le régime hydrodynamique.
2. Lire les données du fichier d'entrainement, extraire les séries relatives aux 4 types de régimes et retracer la carte des régimes avec des symboles différents pour chaque régime. Lire le fichier des données à prédire et ajouter les points sur la carte, afin de vérifier la cohérence des données.
3. Quelle normalisation peut-on appliquer aux données avant de construire le classificateur ? Créer alors un tableau `datanorm` avec ces données normalisées.
4. Lancer l'application "Classification Learner" depuis l'interface Matlab, démarrer une nouvelle session, importer les données du tableau `datanorm`, et appliquer différentes méthode de classification telles que :
   — des arbres de décision "coarse", "medium" et "fine",
   — l'analyse discriminante linéaire,
   — la SVM linéaire, la SVM quadratique,
   — les K plus proches voisins "Fine" et "Medium",
   — d'autres méthodes au choix.
5. Quelle est la meilleure méthode ? Pouvait-on s'attendre à ce résultat ?
6. Générer la fonction de classification automatique et la sauvegarder dans le répertoire du programme principal.
7. Appeler la fonction d'entrainement depuis le programme principal en lui transmettant les données normalisées.
8. Ajouter dans le code la ligne permettant de soumettre au classificateur entrainé les nouvelles données à prédire et stocker ses réponses dans un vecteur.
9. Prédire les étiquettes des nouvelles données.
10. Retracer la figure des régimes en y superposant les nouvelles données avec des symboles correspondant à la première figure : les prédictions sont-elles cohérentes avec les données d'entrainement ?

11.5 Réseaux de neurones

11.5.1 Caractéristiques de perceptrons monocouches

1. Un perceptron monocouche à une entrée (variable x), une couche cachée avec 1 neurone (fonction d'activation linéaire) et une sortie y (activation linéaire) peut-il fournir une sortie y non-linéaire par rapport à l'entrée x ?
2. Un perceptron monocouche à une entrée (variable x), une couche cachée avec 5 neurones (fonction d'activation linéaire) et une sortie y (activation linéaire) peut-il fournir une sortie y non-linéaire par rapport à l'entrée x ?
3. Un perceptron monocouche à une entrée (variable x), une couche cachée avec 1 neurone (fonction d'activation sigmoïde) et une sortie y (activation linéaire) peut-il fournir une sortie y non-linéaire par rapport à l'entrée x ?
4. Combien de paramètres (poids et biais) doivent être entraînés/optimisés dans un perceptron monocouche ayant 3 entrées, une couche cachée à 7 neurones et 2 sorties ?
5. Démontrer que, pour une structure de perceptron monocouche donnée (nombre d'entrées, de neurones et de sorties, et fonction d'activation), il existe plusieurs réglages des poids et biais qui fournissent la même réponse à une entrée donnée.
6. Démontrer que, pour une structure de perceptron monocouche donnée (nombre d'entrées, de neurones et de sorties, et fonction d'activation), il existe plusieurs réglages optimaux des poids et biais (voire une infinité) qui fournissent la même valeur d'erreur entre un ensemble d'observations étiquetées et les prédiction issues de ce "réseau optimal".

11.5.2 Diagnostic des résultats d'entrainement d'un perceptron monocouche

Shalini et al. (2012) ont construit un perceptron monocouche pour établir un modèle en vue de la régulation d'une colonne à distiller. Ce réseau de neurones possède 6 entrées, une couche cachée de 100 neurones et 6 sorties. Lors de la phase d'entrainement, basée sur 18 points de mesure, ils sont parvenus à réduire l'erreur quadratique moyenne à $6.92\ 10^{-27}$ en seulement 5 epochs. Quel diagnostic peut-on établir quant à l'application de la méthode d'entraînement utilisée dans ce cas ? Quel est le risque associé à l'utilisation de réseau de neurones ainsi obtenu ?

11.5.3 Portes logiques

Combien de neurones, N_H, à activation sigmoïde, faut-il inclure dans la couche cachée de perceptrons monocouches $2 - N_H - 1$ à 2 entrées et 1 sortie linéaire pour reproduire chacune des portes logiques OR, XOR et AND dont les tables de vérité sont présentées sur la Figure 11.13 ?

OR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

AND

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

FIGURE 11.13 – Tables de vérité des portes logiques OR, XOR et AND.

11.5.4 Régression de données paraboliques

On considère les données de la Figure 11.14 (gauche). Deux perceptrons monocouches ont été proposés pour régresser ces données (Figures 11.14 (a) et (b)). Chacun de ces réseaux de neurones est-il capable de régresser ces données ? Combien de neurones faudrait-il placer dans la couche cachée de la proposition (a) ? Quelles fonctions d'activation faudrait-il utiliser dans les différents neurones ? Combien de poids et de biais faudrait-il entrainer pour chaque réseau ? Quel architecture de réseau semble préférable ?

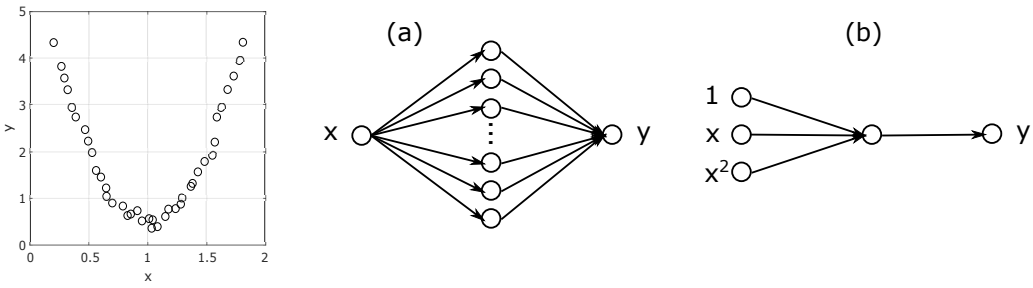


FIGURE 11.14 – Données à régresser et perceptrons monocouches possibles.

11.5.5 Le Mont Fuji

On considère les 60 données de la Figure 11.15 en forme de Mont Fuji. Combien de neurones faudrait-il placer dans la couche cachée à activation sigmoïde d'un perceptron monocouche à sortie linéaire pour régresser les étiquettes y en fonction de x ?

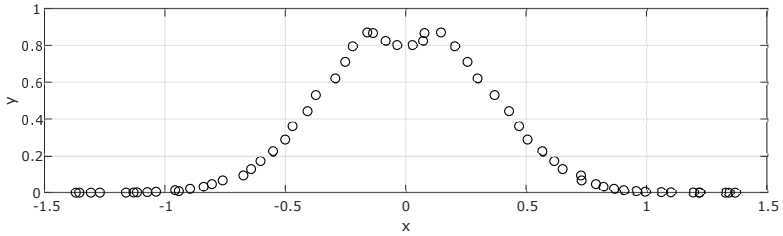


FIGURE 11.15 – Données en forme de Mont Fuji.

Glossaire, abréviations

Glossaire

Apprentissage automatique : Branche de l'intelligence artificielle qui fournit à une machine la capacité à apprendre sans être explicitement programmée dans ce but.

Apprentissage non-supervisé : Situation d'apprentissage automatique où les données ne sont pas étiquetées. Il s'agit donc de découvrir les structures sous-jacentes à ces données. Puisque les données ne sont pas étiquetées, il est impossible à l'algorithme de calculer de façon certaine un score de réussite.

Apprentissage par transfert : Méthode consistant à réutiliser ce qui a été appris sur un problème comme une aide pour en résoudre un autre un peu différent. Pratique courante avec les réseaux convolutionnels profonds, en complétant les couches basses d'un réseau pré-entraîné par de nouvelles couches, puis en effectuant des itérations d'apprentissage sur une nouvelle base d'apprentissage.

Apprentissage profond : Ensemble de méthodes d'apprentissage automatique tentant de modéliser avec un haut niveau d'abstraction des données grâce à des architectures articulées de différentes transformations non linéaires.

Apprentissage supervisé : Tâche d'apprentissage automatique consistant à apprendre une fonction de prédiction à partir d'exemples étiquetés. Les exemples étiquetés constituent une base d'apprentissage, et la fonction de prédiction apprise peut aussi être appelée "hypothèse" ou "modèle". On suppose cette base d'apprentissage représentative d'une population d'échantillons plus large et le but des méthodes d'apprentissage supervisé est de bien généraliser, c'est-à-dire d'apprendre une fonction qui fasse des prédictions correctes sur des données non présentes dans l'ensemble d'apprentissage.

Bottleneck layer : couche d'étranglement (de goulot, d'entonnoir) dans un réseau auto-encodeur.

Bootstrap : Méthode d'échantillonnage statistique permettant de constituer un grand nombre d'échantillons par tirage aléatoire avec remise parmi un ensemble de données. L'application d'une même méthode à ces différents échantillons permet de quantifier la sensibilité du résultat de la méthode à la variabilité des données et de déterminer des intervalles de confiance.

Caractéristique, Decripteur (*feature*) Quantité calculée à partie des entrées brutes des exemples, et qui sera l'entrée du modèle entraîné.

Classification : Sous-type de problème d'apprentissage supervisé, correspondant au cas où la sortie à calculer à partir des entrées est à valeur discrète (et correspond à une classe ou catégorie attribuée).

Classification binaire : Problème d'apprentissage supervisé dans lequel l'espace des étiquettes est binaire (par exemple, 0 ou 1).

Classification multi-classe : Problème d'apprentissage supervisé dans lequel l'espace des étiquettes est discret et fini (par exemple, 0, 1, 2, 3 ou 4).

Clustering : Partitionnement, agrégation, classification. Répartition d'un ensemble de points en sous-ensembles (ou clusters) de points possédant des propriétés analogues.

Compromis biais-variance : Principe statistique visant à construire un modèle avec un niveau de complexité approprié afin de reproduire les tendances générales et significatives d'un ensemble de données sans reproduire la variabilité et l'incertitude de ces données. En apprentissage automatique, ce compromis vise à éviter le sous-apprentissage (biais du modèle) et le sur-apprentissage (variance des données).

Descente de gradient (*gradient descent*) : Algorithme d'optimisation continue permettant de trouver, pour une fonction de coût J convexe l'unique point où J est minimum. Le principe est de calculer en un point initial le vecteur gradient, puis de se déplacer dans la direction opposée, puis d'itérer jusqu'à ce que le gradient devienne nul. Algorithme utilisé pour l'apprentissage des réseaux neuronaux, bien que leur fonction de coût ne soit pas convexe (ce qui nécessite de procéder avec précaution pour éviter de tomber dans un minimum local).

Diagramme de parité : Diagramme à 2 dimensions représentant, pour un ensemble de données, la valeur "expérimentale" de la donnée en abscisse et la valeur calculée par un réseau ou un modèle. Si le modèle est parfait, les points s'alignent sur la première bissectrice du graphe.

Drop-out : Méthode empirique de régularisation applicable aux réseaux neuronaux, et consistant en la désactivation aléatoire de neurones durant une seule itération d'apprentissage.

Data Mining : Fouille de données.

Deep Learning : voir Apprentissage profond.

Ensemble d'apprentissage (*training set*) En apprentissage statistique, ensemble d'exemples utilisé pour appliquer l'algorithme d'apprentissage.

Ensemble de validation (*validation set*) En apprentissage supervisé, ensemble d'exemples étiquetés utilisé pour évaluer les performances (erreurs) des mo-

dèles obtenus par apprentissage, afin de sélectionner de bonnes valeurs pour les hyper-paramètres de l'algorithme, ou bien le meilleur algorithme. L'ensemble de validation doit être disjoint de l'ensemble d'apprentissage.

Ensemble de test (*testing set*) En apprentissage supervisé, ensemble d'exemples étiquetés utilisé pour estimer les performances futures (y compris pour des vecteurs d'entrée jamais vus auparavant) du modèle appris. Il est indispensable, pour que l'estimation soit correcte, que l'ensemble de test soit disjoint non seulement de l'ensemble d'apprentissage qui a servi à apprendre le modèle, mais aussi de l'ensemble de validation qui a permis d'optimiser le choix d'hyperparamètres.

Epoch : Nom générique d'un cycle d'application d'une méthode itérative d'apprentissage statistique, pendant lequel la méthode traite une fois l'ensemble des données d'entrainement.

Erreur empirique (*empirical error*) En apprentissage supervisé, estimation empirique (sur un nombre fini d'exemples étiquetés) de l'erreur d'un modèle. Dans la plupart des algorithmes d'apprentissage, l'erreur empirique sur les exemples d'apprentissage est minimisée, souvent conjointe-ment avec une pénalité de régularisation destinée à contraindre la complexité du modèle.

Erreur de généralisation (*generalization error*) En apprentissage supervisé, c'est l'erreur future du modèle appris sur tous les vecteurs d'entrée possibles. L'objectif de l'apprentissage supervisé est de minimiser cette erreur. Mais celle-ci n'est pas mesurable, on ne peut qu'estimer des erreurs empiriques sur un nombre fini d'exemples existants. Les algorithmes minimisent généralement l'erreur empirique d'apprentissage, souvent augmentée d'un terme de pénalisation pour contraindre le modèle à bien généraliser.

Etiquette : Grandeur, associée à une donnée, que doit prédire le système supervisé.

Evanouissement du gradient : Problème rencontré lors de l'apprentissage des réseaux de neurones par les méthodes de rétropropagation, où les fonctions d'activation non-linéaires (fonctions à seuil, Heaviside, ReLU) induisent une annulation locale des dérivées et une nullité du gradient par propagation.

Fléau de la dimension : Terme introduit en 1961 par Richard Bellman pour désigner les difficultés rencontrées par les méthodes de traitement de données dans des espaces de très grande dimension. Cette "malédiction de la dimensionnalité" se comprend bien en calculant la proportion de "volume" qu'occupe une boule dans son enveloppe cubique (Figure 12.1) : cette proportion de volume représente la fraction des données qui se trouvent à une distance inférieure à 1 du centre. En 1D, cette proportion vaut 1 (tout le domaine est à une distance inférieure à 1 du centre), elle descend à $\frac{\pi}{4} = 0,78$ en 2D, puis à $\frac{\pi}{6} = 0,52$ en 3D. Cette fraction décroit lorsque la dimension augmente : en dimension 8, l'hypersphère unitaire ne représente que 1,5 % du volume de l'hypercube unitaire. Cela signifie qu'il y a très peu de données qui se trouvent à proximité du centre. En fait, plus la dimension augmente, plus les données semblent éloignées les unes des autres : la notion de voisinage disparait rapidement avec l'augmentation de la dimension du problème.

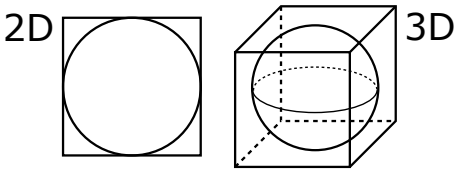


FIGURE 12.1 – Boules 2D et 3D dans leurs enveloppes cubiques.

Hidden Layer : Couche cachée.

Hivers de l'IA : On appelle "hivers" de l'IA deux périodes de l'histoire de son développement pendant lesquelles ces technologies ont semblé atteindre leurs limites ou se sont confrontés à des impasses méthodologiques. Le premier hiver s'est produit à la fin de la période des systèmes experts au milieu des années 90 : le développement de systèmes experts crédibles s'est avéré très chronophage et coûteux, en développement et en maintenance, ce qui a été rédhibitoire à leur utilisation. Le second hiver de l'IA, au début des années 2000, a marqué la fin des premiers développements des réseaux de neurones (monocouches) : malgré des résultats impressionnants, ils échouaient à résoudre des problèmes complexes comme l'analyse d'images. La communauté avait identifié le besoin de couches multiples, mais les algorithmes ne permettaient pas d'entraîner de tels réseaux : ce second hiver s'achève lorsque ces algorithmes de rétropropagation sont découverts, ce qui a donné lieu à l'émergence du Deep Learning (Venkatasubramanian, 2019).

Ian Goodfellow : Chercheur en intelligence artificielle, auprès de Google (2014-2019) et Apple (2019 -...). Auteur du livre de référence "Deep Learning" avec Yoshua Bengio et Aaron Courville, les directeurs de sa thèse soutenue en 2014. Inventeur en 2014 des réseaux antagonistes génératifs.

Iris de Fisher : Jeu de données multivariées présenté en 1936 par Ronald Fisher dans son article "*The use of multiple measurements in taxonomic problems*" comme un exemple d'application de l'analyse discriminante linéaire. Ces données sont couramment utilisées comme cas d'école en apprentissage automatique, notamment en clustering.

Machine Learning : voir Apprentissage automatique.

Méthode ensembliste : Catégorie d'algorithmes d'apprentissage dont le principe commun est de combiner plusieurs sous-modèles simples, et dont la sortie est obtenue par un vote ou une moyenne (avec ou sans pondérations) des sorties des sous-modèles. Les Forêts Aléatoires et le boosting appartiennent à cette catégorie d'algorithmes.

Modèle génératif : Algorithme générant des échan-

tillons suivant une probabilité $p(x)$ donnée de façon explicite ou estimée de façon implicite à partir des données d'entraînement x_1, ..., x_N.

Over-fitting : Sur-apprentissage.

Outlier : Point aberrant.

Parity Plot : voir Diagramme de parité.

Partitionnement (ou *Clustering*) : Problème d'apprentissage non-supervisé consistant à diviser un ensemble de données en différents "paquets" homogènes, au sens où les données de chaque sous-ensemble partagent des caractéristiques communes. On définit des critères de proximité (similarité informatique) en introduisant des mesures de distance entre objets.

Pooling : Opération couramment utilisée en Deep Learning, notamment dans les réseaux convolutifs en traitement d'images, pour réduire progressivement la taille des couches de neurones, qui consiste à remplacer la sortie d'un groupe de neurones par une statistique relative à ce groupe. Le *max pooling* consiste ainsi à ne retenir que la valeur maximale d'un ensemble de neurones dans un voisinnage donné.

Principe de parcimonie : voir "Rasoir d'Ockham".

Pruning : (traduction : élagage) Opération consistant à supprimer des neurones ou des liaisons "inactives" dans un réseau de neurones pour en réduire le nombre de paramètres sans perte en qualité.

Rasoir d'Ockham : Principe de raisonnement philosophique entrant dans les concepts de rationalisme et de nominalisme. Le terme vient de "raser" qui, en philosophie, signifie « éliminer des explications improbables d'un phénomène » et du philosophe du XIV^{eme} siècle Guillaume d'Ockham. Également appelé principe de simplicité, principe d'économie ou principe de parcimonie, il peut se formuler comme suit :

> *Pluralitas non est ponenda sine necessitate*
> (les multiples ne doivent pas être utilisés sans nécessité)

Une formulation plus moderne est que « les hypothèses suffisantes les plus simples doivent être préférées ». C'est un principe heuristique fondamental en science, sans être un résultat scientifique. En langage courant, le rasoir d'Ockham pourrait sa traduire par "Pourquoi faire compliqué quand on peut faire simple ?".

Réduction de dimensionnalité : Problème d'apprentissage non-supervisé consistant à rechercher un sous-espace de dimension plus faible que l'espace des données, mais sur lequel les projections des données conservent des caractéristiques utiles.

Régression : Problème d'apprentissage supervisé dans lequel l'espace des étiquettes est $\mathbb{R}$.

Régularisation : La régularisation est une technique visant à forcer un modèle à rester aussi simple que possible. En apprentissage supervisé, on souhaite régulariser le modèle durant l'apprentissage afin qu'il généralise le mieux possible (au lieu de simplement apprendre par coeur les exemples d'apprentissage). Pour cela, une approche fréquemment utilisée est d'ajouter à la fonction de coût, en plus de l'erreur empirique d'apprentissage, un terme de régularisation qui pénalise les modèles plus complexes (car ils généralisent moins bien).

Rétropropagation : Méthode permettant de calculer le gradient de l'erreur pour chaque neurone d'un réseau de neurones, depuis la couche de sortie vers la couche d'entrée, et de corriger les poids synaptiques du réseau et les biais, afin de minimiser l'erreur d'entrainement en sortie.

Réseau de neurones à propagation avant : réseau de neurones artificiels acyclique, se distinguant ainsi des réseaux de neurones récurrents. Le plus connu est le perceptron multicouche qui est une extension du premier réseau de neurones artificiel, le perceptron inventé en 1957 par Frank Rosenblatt. Le réseau de neurones à propagation avant était le premier et le plus simple type de réseau neuronal artificiel conçu. Dans ce réseau, l'information ne se déplace que dans une seule direction, vers l'avant, à partir des noeuds d'entrée, en passant par les couches cachées (le cas échéant) et vers les noeuds de sortie.

Réseaux convolutifs / convolutionnels : Type de réseau de neurones très utilisé en Deep Learning, notamment en traitement d'images. Contrairement aux réseaux classiques où chaque neurone d'une couche est connecté à tous les neurones de la couche précédente, les neurones d'une couche d'un réseau à convolution ne sont connectés qu'à un groupe de neurones "voisins". En traitement d'images, chaque neurone correspond à un pixel : la convolution est donc une trasformation de ce pixel en fonction de ses pixels voisins, une sorte de filtre. Cette méthode permet de détecter des structures locales dans l'image (frontière, contraste, etc.).

Réseaux évolutionnaires : réseaux pour lesquels la méthode d'entraînement ne joue pas que sur les poids et biais, mais aussi sur les hyper-paramètres (nombre de couches, nombre de neurones par couche, fonction d'activation).

Richard Bellman (1920-1984) : Mathématicien américain reconnu pour ses contributions à la théorie de la décision et à la théorie du contrôle optimal, notamment par la création et les applications de la programmation dynamique (Figure 12.2). L'algorithme de Ford-Bellman permet de déterminer les plus courts chemins à partir d'un sommet dans un graphe orienté pondéré quelconque. Bellman a lancé l'expression "fléau de la dimension".

Ronald Aylmer Fisher (Sir) (1890-1962) : Biologiste et statisticien britannique (Figure 12.3), il est considéré comme l'un des fondateurs des statistiques modernes, par l'introduction de nombreux concepts clés tels que le maximum de vraisemblance, l'information de Fisher, l'analyse de la variance et les plans d'expériences.

Stéphane Mallat : Professeur en Sciences des données au Collège de France.

Supervised learning : voir Apprentissage supervisé.



FIGURE 12.2 – Richard Bellman.



FIGURE 12.3 – Sir Ronald Fisher en 1913.

Sur-apprentissage (Sur-ajustement) : Caractéristique 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. Le sur-apprentissage est en général provoqué par un mauvais dimensionnement de la structure utilisée pour classifier ou faire la régression.

Teuvo Kalevi Kohonen : Chercheur finlandais, professeur émérite auprès de l'académie de Finlande, reconnu pour son travail sur les réseaux de neurones. Inventeur des cartes auto-adaptatives, aussi appelées réseaux de Kohonen.

Under-fitting : Sous-apprentissage.

Unsupervised learning : voir Apprentissage non-supervisé.

Validation croisée (*cross-validation*) Méthode permettant d'évaluer de façon robuste, même si le nombre total d'exemples disponibles est restreint, l'erreur de validation (afin de choisir un algorithme ou les valeurs de ses hyper-paramètres). Son principe consiste à découper l'ensemble des exemples d'apprentissage en K sous-parties de tailles égales et tirées aléatoirement, puis à entraîner sur $(K-1)/K$ et évaluer sur le reste, et moyenner les K erreurs de validation obtenues ainsi.

Variables latentes : Variables représentatives résiduelles obtenues dans le goulot d'étranglement d'un réseau auto-encodeur.

Yann Le Cun : Chercheur en intelligence artificielle et vision artificielle (robotique), considéré comme l'un des inventeurs de l'apprentissage profond. Prix Turing 2019. Titulaire de la chaire « Informatique et sciences numériques » du Collège de France. En 2018, il devient chef de l'IA chez Facebook.

Abréviations

AAD *Average Absolute Deviation*, voir Tableau 7.2.

ACP : Analyse en Composantes Principales

ANN *Artificial Neural Network*

ANOVA *Analysis of Variance*, voir cours de Méthodes Statistiques, ENSIC, I_2C 2A.

BANN *Bootstrap Aggregated Neural Network.*

CNN *Convolutional Neural Network*

CPU *Central Processing Unit*, Processeur Central d'un ordinateur.

DBSCAN *Density based clustering of application with noise*, Partitionnement basé sur la densité.

DDA *Diagonal Discriminant Analysis.*

DPLS *Discriminant Partial Least Squares.*

EM *Expectation Maximization.*

FDA *Fisher Discriminant Analysis.*

FFNN *Feed-Forward Neural Network*, voir "Réseau de neurones à propagation avant".

FIS *Fuzzy Inference System.*

GA *Genetic Algorithm*, Algorithme génétique.

GAN *Generative Adversarial Network*, Réseau génératif adverse.

GMM *Gaussian Mixture Model.*

GPU *Graphics Processing Unit*, Processeur graphique.

HDFS *Hadoop Distributed File System.*

HMM *Hidden Markov Model.*

IoT *Internet of Things.*

IRN *Internally Recurrent Net.*

LASSO *Least Absolute Shrinkage and Selection Operator.*

LDA *Linear Discriminant Analysis.*

LLE *Locally linear Embedding*, Plongement localement linéaire.

LM Levenbergh-Marquardt.

LSTM *Long Short-Term Memory.*

MAE *Mean Absolute Error*, voir Tableau 7.2.

MARE *Maximum Absolute Relative Error*, voir Tableau 7.2.

MCCV *Monte-Carlo Cross Validation.*

MCL *Markov Clustering.*

MDS *MultiDimensional Scaling*, Positionnement multidimensionnel.

MLP *Multi-Layer Perceptron.*

MLRA *Multiple Linear Regression Analysis.*

MMC Modèle de Markov Caché.

MPC *Model Predictive Control*, Commande prédictive basée sur modèle.

MRE *Mean Relative Error*, voir Tableau 7.2.

MSE *Mean Squared Error*, voir Tableau 7.2.

PCR *Principal Component Regression.*

PLS *Partial Least Squares.*

PMC Perceptron Multi-Couche.

PSO *Particle Swarm Optimization*, Optimisation par essaim de particules.

QDA *Quadratic Discriminant Analysis.*

RF *Random Forest.*

RMSE *Root-Mean Squared Error*, voir Tableau 7.2.

RMSLE *Root-Mean Squared Log Error*, voir Tableau 7.2.

RNG *Random Number Generator.*

SDA *Shrinkage Discriminant Analysis.*

SGD *Stochastic Gradient Descent.*

SOM *Self-Organizing Map.*

SSPCR *Semi-Supervised Principal Component Regression.*

SVD *Singular Value Decomposition.*

SVM *Support Vector Machine*, Séparateur à Vaste Marge.

SWA *Stochastic Weight Averaging.*

t-SNE *t-distributed Stochastic Neighbor Embedding.*

Bibliographie

[1] Aggarwal C. C., *Data mining – the textbook*, Ed. Springer, 2015.

[2] Aghaji, M. Z., Fernandez, M., boyd, P. G., Daff, T. D., Woo, T. K., *Quantitative Structure-Property Relationship models for recognizing Metal Organic Frameworks (MOFs) with high CO_2 working capacity and CO_2/CH_4 selectivity for methane purification*, Eur. J. Inorg. chem., 4505-4511, 2016.

[3] Alabi, S. B., Williamson, C. J., *Neural network-based model for joint prediction of the Newtonian and nonNewtonian viscosities of black liquor*, Int. J. Chem. Eng. App., 6, 3, 195-200, 2015.

[4] Al-Hemiri, A. A., Salih, S. A., *Prediction of mass transfer coefficient in bubble column using artificial neural network*, Journal of Engineering, 13, 2, 2007.

[5] Alves, R. M. B., Nascimento, C. A. O., *Gross errors detection of industrial data by neural networks and cluster techniques*, Brazilian J. Chem. Eng., 19, 4, 483-489, 2002.

[6] Anitha, M., Singh, H., *Artificial neural network simulation of rare earths solvent extraction equilibrium data*, Desalination, 232, 59-70, 2008.

[7] Atasoy, I., Yuceer, M., Ulker, E. O., Berber, R., *Neural network based control of the acrylonitrile polymerization process*, Chem. Eng. Technol., 30, 11, 1525-1531, 2007.

[8] Attarian Shandiz, M., Gauvin, R., *Application of machine learning methods for the prediction of crystal system of cathode materials in lithium-ion batteries*, Computational Materials Science, 117, 2770-278, 2016.

[9] Azencott, C.-A., *Introduction au machine learning*, Ed. Dunod, 2018.

[10] Azhar, S., Rahman, A.B., *Application of artificial neural network in fault detection study of batch esterification process*, Int. J. Eng. Tech., IJET-IJENS, 10, 3, 36-39, 2010.

[11] Basile, A., Curcio, S., Bagnato, G., Liguori, S., Jokar, S.M., Iulianelli, A., *Water gas shift reaction in membrane reactors : theoretical investigation by artificial neural networks model and experimental validation*, Int. J. Hydrogen Energy, 40, 5897-5906, 2015.

[12] Bollas, G. M., Papadokonstadakis, S., Michalopoulos, J., Arampatzis, G., Lappas, A. A., Vasalos, I. A., Lygeros, A., *Using hybrid neural networks in scalinp up an FCC model from a pilot plant to an industrial plant*, Chem. Eng. Proc., 42, 697-713, 2003.

[13] Boozarjomehry, R. B., Abdolahi, F., Moosavian, M. A., *Characterization of basic properties for pure substances and petroleum fractions by neural network*, Fluid Phase Equilibria, 231, 188-196, 2005.

[14] Biyanto, T. R., Widjiantoro, B. L., Abo Jabal, A., Budiati, T., *Artificial neural network based modeling and controlling of distillation column system*, Int. J. Eng. Science Tech., 2, 6, 177-188, 2010.

[15] Chen, L., Gasteiger, J., *Knowledge discovery in reaction databases : landscaping organic reactions by a self-organizing neural network*, J. Am. Chem. Soc., 119, 4033-4042, 1997.

[16] Chiang, L. H., Russell, E. L., Braatz, R. D., *Fault diagnosis in chemical processes using Fisher discriminant analysis, discriminant partial least squares, and principal component analysis*, Chemometrics and Intelligent Systems, 50, 243-252, 2000.

[17] Chouai, A., Cabassud, M., Le Lann, M. V., Gourdon, C., Casamatta, G., *Use of neural networks for liquid-liquid extraction column modelling : an experimental study*, Chem. Eng. Proc., 39, 171-180, 2000.

[18] Chouai, A., Laugier, S., Richon, D., *Modeling of thermodynamic properties using neural networks - application to refrigerants*, Fluid Phase Equilibria, 199, 53-62, 2002.

[19] Coley, C. W., Barzilay, R., Kaakkola, T. S., Green, W. H., Jensen, K. F., textitPrediction of organic reaction outcomes using machine learning, ACS Cent. Sci., 3, 434-443, 2017.

[20] Corma, A., Serra, J. M., Serna, P., Moliner, M., *Integrating high-throughput characterization into combinatorial heterogeneous catalysis : unsupervised construction of quantitative structure/property relationship models*, J. Catalysis, 232, 335-341, 2005.

[21] Cybenko, G., *Approximations by superpositions of sigmoidal functions*, Mathematics of Control, Signals, and Systems, 2, 4, 303–314, 1989.

[22] Dal-Pastro, F., Facco, P., Zamprogna, E., Bezzo, F., Barolo, M., *Model-based approach to the design and scale-up of wheat milling operations - proof of concept*, Food and Bioproducts Processing, 106, 127-136, 2017.

[23] Eilermann, M., Post, C., Schwarz, D., Leufke, S., Schembecker, G., Bramsiepe, C., *Generation of an equipment database for heat exchangers by cluster analysis of industrial applications*, Chem. Eng. Sci., 167, 278-287, 2017.

[24] Eslamloueyan, R., Khademi, M. H., *Using artificial networks for estimation of thermal conductivity of binary gaseous mixtures*, J. Chem. Eng. Data, 54, 922-932, 2009.

[25] Eslamloueyan, R., Khademi, M. H., Mazinani, S., *Using a multilayer perceptron network for thermal conductivity prediction of aqueous electrolyte solutions*, Ind. Eng. Chem. Res., 50, 4050-4056, 2011.

[26] Espinasse, B., Bellot, P., *Introduction au Big Data - Opportunités, stockage et analyse des mégadonnées*, Techniques de l'Ingénieur, H6040, 2017.

[27] Essiet, I., Sun, Y., Wang, Z., *Big Data analysis for gas sensor using convolutional neural network and ensemble of evolutionary algorithms*, Procedia Manufacturing, 35, 629-634, 2019.

[28] Fan, J. Y., Nikolaou, M., White, R. E., *An approach to fault diagnosis of chemical processes via neural networks*, AIChE Journal, 39, 1, 82-88, 1993.

[29] Fernandez de Canete, J., Gonzalez-Perez, S., del Saz-Orozco, P., *Artificial neural networks for identification and control of a lab-scale distillation column using Labview*, World Academy of Science, engineering and Technology, 47, 64-69, 2008.

[30] Fernandez, M., Barnard, A. S., *geometrical properties can predict CO_2 and N_2 adsorption performance of metal-organic frameworks (MOFs) at low pressure*, ACS Comb. Sci., 18, 243-252, 2016.

[31] Fernandez, M., Barron, H., Barnard, A. S., *Artificial neural network analysis of the catalytic efficiency of platinum nanoparticles*, RSC advances, 7, 48962-48971, 2017.

[32] Ganguly, S., *Prediction of VLE data using radial basis function*, Comp. Chem. Eng., 27, 1445-1454, 2003.

[33] Gasteiger, J., Zupan, J., *Natural networks in chemistry*, Angew. Chem. Int. Ed. Engl., 32, 503-527, 1993.

[34] Goldsmith, B.R., Esterhuizen, J., Liu, J.X., Bartel, C.,J., Sutton, C., *Machine learning for heterogeneous catalyst design and discovery*, AIChE Journal, 64, 7, 2311-2323, 2018.

[35] Goodfellow, I., *L'apprentissage profond*, Massot Editions, 2018.

[36] Gunesoglu, S., Kaplangiray, B., *Applying the artificial neural network to predict the thermal properties of knitted fabrics*, Fibres and Textiles, 1, 41-44, 2019.

[37] Gupta, A., *Introduction to deep learning : part 1*, Chem. Eng. Prog., 22-29, Juin 2018.

[38] Hachemaoui, A., Belhamel, K., *an artificial neural network approach for the prediction of extraction performance of emulsion liquid membrane*, J. Chem. Eng. Proc. Tech., 8, 4, 1000356, 2017.

[39] Himmelblau, D. M., Barker, R. W., Suewatanakul, W., *Fault classification with the aid of artificial neural networks*, IFAC Fault Detection, Supervision and Safety for Technical Processes, 541-545, 1991.

[40] Himmelblau, D. M., *Applications of artificial neural networks in chemical engineering*, Korean J. Chem. Eng., 17, 4, 373-392, 2000.

[41] Hoskins, J. c., Kaliyur, K. M., Himmalblau, D. M., *Fault diagnosis in complex chemical plants using artificial neural networks*, AIChE Journal, 37, 1, 137-141, 1991.

[42] Ihme, M., Marsden, A. L., Pitsch, H., *On the optimization of arttificiel neural networks for application to the approximation of chemical systems*, Center for Turbulence Reasear Annual Research Briefs 2006, 105-118, 2006.

[43] Iliuta, M. C., Iliuta, I., Larachi, F., *Vapour-liquid equilibrium data analysis for mixed solvent-electrolyte systems using neural network models*, Chem. Eng. Sci., 55, 2813-2825, 2000.

[44] Issanchou, S., Gauchi, J. P., *Computer-aided optimal designs for improving neural network generalization*, Neural Networks, 21, 945-950, 2008.

[45] Khalil Arya, F., Ayati, B., *Application of artificial neural networks for predicting COD removal eficiencies of rotating disks and packed-cage RBCs in treating hydroquinone*, IJST, Transactions of Civil Eng., 37, C2, 325-336, 2013.

[46] Khataee, A. R., Dehghan, G., Zarei, M., Ebadi, E., Pourhassan, M., *Neural network modeling of biotreatment of triphenylmethane dye solution by a green macroalgae*, Chem. Eng. Res. Des., 89, 172-178, 2011.

[47] Kramer, M. A., *Nonlinear Principal Component Analysis using Autoassociative neural networks*, AIChE Journal, 37, 2, 233-243, 1991.

[48] Kurt, H., Kayfeci, M., *Prediction of thermal conductivity of ethylene glycol-water solutions by using artificial neural networks*, Applied Energy, 86, 2244-2248, 2009.

[49] Leite M. S., Santos, M., A., Costa, E. M. F., Balieiro, A., Lima, A. S., Sanchez, O. L., Soares, C. M. F., *Modeling of milk lactose removal by column adsorption using artificial neural networks : MLP and RBF*, Chem. Ind. Chem. Eng., 25, 4, 369-382, 2019.

[50] Li, H., Chen, F., Cheng, K., Zhao, Z., Yang, D., *Prediction of zeta potential of decomposed peat via machine learning : comparative study of Support Vector Machine and Artificial Neural Networks*, Int. J. Electrochem. Sci., 10, 6044-6056, 2015.

[51] Li, H., Zhang, Z., Liu, Z., *Application of artificial neural networks for catalysis : a review*, Catalysts, 7, 306, 2017.

[52] Li, H., Zhang, Z., textitMining the intrinsic trends of CO2 solubility in blended solutions, J. CO_2 Utilization, 26, 496-502, 2018.

[53] Li, H., Zhang, Z., Zhao, Z. Z., *Data-mining for processes in chemistry, materials and engineering*, Processes, 7, 151, 2019.

[54] Lisa, G., Curteanu, S., *Artificial neural network modeling for density of some binary systems*, Scientific Study and Research, 6, 1, 65-79, 2005.

[55] Loney, N. W., Simon, L., Gao, L., *Trends in the applications of neural networks in chemical process modelling*, Proc. Indian Natn Sci. Acad., 69, A, 3-4, 285-299, 2003.

[56] Maltarollo, V. G., Honorio, K. M., Da Silva, A. B. F., *Applications of artificial neural networks in chemical problems*, http ://dx.doi.org/10.5772/51275.

[57] Spelling, M., Glotzer, S. C., *Machine learning for crystal identification and discovery*, AIChE Journal, 64, 6, 2198-2206, 2018.

[58] Moghadassi, A., Parvizian, F., Hosseini, S. M., *Application of artificial neural network for prediction of liquid viscosity*, Indian Chemical Engineer, 52, 1, 37-48, 2010.

[59] Moutarde, F., *Apprentissage statistique supervisé*, Techniques de l'Ingénieur, H 5010, 2019.

[60] Mujtaba, I. M., Aziz, N., Hussain, M. A., *Neural network based modelling and control in batch reactor*, Trans. IChemE, Part A, 84, A8, 635-644, 2006.

[61] Nabil, T., Le Moullec, Y., Le Coz, A., *Machine learning based design of a supercritical CO_2 concentrating solar power plant*, 3^{rd} European supercritical CO_2 Conference, Paris, France, 19-20 septembre, 2019.

[62] Nandy, A., Duan, C., Janet, J. P., Gugler, S., Kulik, H. J., *Strategies and software for machine learning accelerated discovery in transition metal chemistry*, Ind. Eng. Chem. Res., 57, 13973-13986, 2018.

[63] NAscimento, C. A. O., Giudici, R., Guardani, R., *Neural network based approach for optimization of industrial chemical processes*, Comp. Chem. Eng., 24, 2303-2314, 2000.

[64] Oguntade, T. I., Ita, C. S., Sanmi, O., Oyekunle, D. T., *A binary mixture of sesame and castor oil as an ecofriendly corrosion inhibitor of mild steel in crude oil*, The Open Chem. Eng. J., 14, 25-35, 2020.

[65] Onel, M., Kieslich, C. A., Guzman, Y. A., Floudas, C. A., Pistikopoulos, E. N., "*Big data approach to batch process monitoring : simultaneous fault detection and diagnosis using nonlinear support vector machne-based feature selection*, Comp. Chem. Eng., 115, 46-63, 2018.

[66] Otero, F., *Use of neural networks in process engineering - thermodynamics, diffusion, process control and simulation applications*, Ciencia, Tecnologia y Futuro, 1, 4, 49-64, 1998.

[67] Pandharipande, S. L., Nagdive, A., Moharkar, Y., *Modeling of liquid-liquid extraction in spray column using artificial neural network*, Int. J. Scientific and Research Publications, 2, 6, 2012.

[68] Paoli, C., Voyant, C., Muselli, M., Nivet, M.L., *Forecasting of preprocessed daily solar radiation time series using neural networks*,

[69] Patnaik, P. R., *Applications of neural networks to recevery of biologicial products*, Biotechnology Advances, 19, 477-488, 1999.

[70] Plehiers, P.P., Symoens, S. H., Amghizar, I., Marin, G. B., Stevens, C. V., Van Geem, K. M., *Artificial intelligence in steam cracking modeling : a deep learning algorithm for detailed effluent prediction*, Engineering, 2019.

[71] Prakaxh Maran, J., Priya, B., *Comparison of response surface methodology and artificial neural network approach towards efficient ultrasound-assisted biodiesel production from muskmelon oil*, Ultrasonics Sonochemistry, 23, 192-200, 2015.

[72] Ras, E. J., McKay, B., Rothenberg, G., *Understanding catalytic biomass conversion through data mining*, Top. Catal., 53, 1202-1208, 2010.

[73] Razavi, M. A., Mortazavi, A., Mousavi, M., *Dynamic modeling of milk ultrafiltration by artificial neural network*, J. Memb. Sci., 220, 47-58, 2003.

[74] Rosli, M. N., Aziz, N., *Review of neural network modelling of cracking process*, IOP Conf. Series : Materials Science and Engineering, 162, 012016, 2017.

[75] Rothenberg, G., *Data mining in catalysis : separating knowledge from garbage*, Catalysis Today, 137, 2-10, 2008.

[76] Sabzevari, S., Moosavi, M., *Density prediction of liquide alkali metals and their mixtures using an artificial neural network method over the whole liquid range*, Fluid Phase Equilibria, 361, 135-142, 2014.

[77] Santos, R. R. C., Santos, B. F., Fileti, A. M. F., Da Silva, F. V., Zemp, R. J., *Application of artificial neural networks in an experimental batch reactor of styrene polymerization for predictive model development*, Chem. Eng. Transactions, 32, 1399-1404, 2013.

[78] Sauvalle, B., *Apprentissage statistique non supervisé*, Techniques de l'Ingénieur, H5012, 2020.

[79] Shalini, J., Sunil Kumar, J., Akhila Swathanthra, P., *Steady-state modeling of chemical systems by using nerual networks*, Int. J. Eng. Res. & Tech., 1, 6, 2012.

[80] Simon, C.M., Mercado, R., Schnell, S.K., Smit, B., aranczyk, M., *What are the best materials to separate a Xenon/Krypton mixture ?*, Chem. Mater., 27, 4459-4475, 2015.

[81] Teng, S. Y., Masa, V., Stehlik, P., Lam, H. L., *Deep learning approach for industrial process improvement*, Chem. Eng. Transactions, 76, 487-492, 2019.

[82] Tufféry, S., *Big Data, Machine Learning et apprentissage profond*, Ed. Technip, 2019.

[83] Vasickaninova, A., Bakosova, M., Meszaros, A., *Control of heat exchangers in series using neural network predictive controllers*, Acta Chimica Slovaca, 13, 1, 41-48, 2020.

[84] Venkatasubramanian, V., Chan, K., *A neural network methodology for process fault diagnosis*, AIChE J., 35, 12, 1993-2002, 1989.

[85] Venkatasubramanian, V., *The promise of artificial intelligence in chemical engineering : is it here, finally ?*, AIChE Journal, 65, 2, 466-478, 2019.

[86] Voyant, C., Notton, G., Kalogirou, S., Nivet, M.L., Paoli, C., Motte, F., Fouilloy, A., *Machine learning methods for solar radiation forescasting : a review*, Renewable Energy, 105, 569-582, 2017.

[87] Walter, E., Pronzato, L., *Identification de modèles paramétriques à partir de données expérimentales*, Ed. Masson, Paris, 1994.

[88] Wang W. C., Li, J., Wang, X., *Recurrent neural network-based model predictive control for continuous pharmaceutical manufacturing*, arXiv : 1807.09556v1 [cs.SY], 2018.

[89] Yao, L., Ge, Z., *Big data quality prediction in the process industry : a distributed parallel modeling framework*, Journal of Process Control, 68, 1-13, 2018.

[90] Yoo, Y., *Data-driven fault detection process using correlation based clustering*, Computers in Industry, 122, 103279, 2020.

[91] Zhang, Y., Zhang, R. Z., Wang, Y., Guo, H., Zhong, R. Y. Qu, T., Li, Z., *Big data driven decision-making for batch-based production systems*, 11^{th} CIRP Conference on Industrial Product-Service Systems, Procedia CIRP 83, 814-818, 2019.

[92] Zhou, T., Song, Z., Sundmacher, K., *Big Data creates new opportunities for materials research : a review on methods and applications of machine learning for materials design*, Engineering, 2019.

[93] Zivkovic, Z., Mihajlovic, I., Nikolic, D., *Artificial neural network method applied on the nonlinear multivariate problems*, Serbian Journal of Management, 4, 2, 143-155, 2009.

Index

ACID, 3
ACP, 33, 95
Adaboost, 56
Analyse d'images, 79
Analyse discriminante, 96
Analyse discriminante de Fisher, 40
Analyse en Composantes Principales, 33, 95
Analyse hiérarchique de clusters, 25
Apprentissage automatique, 1
Apprentissage non-supervisé, 5
Apprentissage profond, 1
Apprentissage supervisé, 5
Arbres de décision, 51, 96
Astuce du noyau, 51
Average Absolute Deviation, 63

Bagging, 55
Bayes, 57
BFGS, 71
Big Data, 1
Boosting, 56
Bootstrap, 20, 55

Cartes auto-adaptatives, 35, 92, 96, 98
Cartes auto-organisées, 35
Cartes de Kohonen, 92, 98
Cartes topologiques, 35
Centroïde, 22
Classification, 5, 96
Cloud computing, 3
Clustering, 5, 90
Clustering hiérarchique, 25
Clustering par densité, 26
Coefficient de corrélation, 9
Coefficient de détermination, 13
Coefficient de silhouette, 23
Compromis biais-complexité, 13
Compromis biais-variance, 13
Convolution, 79
Couche dense, 66
Courbe ROC, 46

Data Mining, 1
DBSCAN, 27, 91
Decision tree, 51
Deep Learning, 1, 3
Dendrogramme, 25, 91
Descente de gradient, 71
Descripteurs, 18
Diagramme de parité, 11
Données d'entrainement, 18
Données de test, 18
Données de validation, 18
Données manquantes, 15
Données massives, 1
Données saturées, 15
DropConnect, 72
Dropout, 72
Décodeur, 37, 102

Encodeur, 37, 102
Environnements de développement, 87
Erreur de généralisation, 18
Erreur empirique, 18
Erreurs de première et seconde espèce, 46
Explicabilité, 72

Factorisation en matrices non-négatives, 40
FDA, 40
Fichiers, 90
Fisher Discriminant Analysis, 40
Fletcher-Reeves, 71
Fonction d'erreur, 63
Fonction de coût, 63
Fonction de transfert, 63
Forêts aléatoires, 55
Fouille de données, 1

Gradient conjugué, 71
Généralisation, 18

Hadoop, 3
Homogénéité d'un cluster, 22
Hyper-paramètres, 64

Inférence bayesienne, 57
Intelligence Artificielle, 1, 3
Interprétabilité, 72

K plus proches voisins, 96
K-moyennes, 23
kNN, 47, 96

Laplacian Eigenmaps, 41
Leave-one-out, 19
Levenbergh-Marquardt, 71
LLE, 41
Local Linear Embedding, 41
Logiciels, 87
Loi de Bayes, 57
LSTM, 81

Machine Learning, 1, 3

Machine à vecteur de support, 48
Machines de Boltzmann restreintes, 38
MapReduce, 3
Marge d'un séparateur, 48
Matlab, 88
Matrice de confusion, 45
Matrice de corrélation, 9
Matrice de covariance, 8
Matrice de projections, 15
Mean Relative Error, 63
Mean Squared Error, 63
Mean-shift clustering, 29
Modèles de Markov, 30
Modèles de mélange gaussiens, 28
Moindres carrés partiels, 39
Multi-Layer Perceptron (MLP), 66
Médoïde, 22
Méthode des K plus proches voisins, 47
Méthode des K-moyennes, 23, 90
Méthodes ensemblistes, 55, 96
Méthodes spectrales, 30

NewSQL, 3
Normalisation de variables, 17
Normalisation min-max, 17
NoSQL, 3

Paramètres, 64
Partitionnement, 90
PCA, 33
Plongement local linéaire, 41
Plongement spectral, 41
Polak-Ribière, 71
Pooling, 80
Positionnement multi-dimensionnel, 38
Powell-Beale, 71
Principe des 3V et 5V, 2
Projections aléatoires, 41
Propagation avant, 66
Propagation d'affinités, 28
Pruning, 72
Python, 87

Quasi-Newton, 71

R, 87
Randon forest, 55
Représentation latente, 37
Réduction de dimensionnalité, 5, 33, 95
Régression, 5
Régression linéaire multiple, 10
Réseau auto-encodeur, 37, 102
Réseau convolutionnel, 79
Réseau de Elman, 81
Réseaux auto-encodeurs, 82
Réseaux de Kohonen, 35
Réseaux de neurones, 53
Réseaux multicouches, 66
Réseaux récurrents, 81
Rétropropagation, 71

Scikit Learn, 87
Sensibilité, 46
Silhouette, 23
Spécificité, 46
SQL, 3
Support Vector Machines, 96
Surface AUC, 46
SVM, 96
SVM - Support Vector Machine, 48
SVM - Séparateur à Vaste Marge, 48
Séparabilité, 22
Séparateurs à Vastes Marges, 96

t-SNE, 39
Tensorflow, 87

Validation croisée, 19
Variable centrée réduite, 17
Variables latentes, 37
Visualisation, 15
Voisinage, 27

Weka, 87