L'Apprentissage artificiel au service du profilage des données

Marc Chevallier

UNIVERSITÉ PARIS XIII - SORBONNE PARIS NORD
École Doctorale Sciences, Technologies, Santé Galilée

L'Apprentissage Artificiel au service du profilage des données

THÈSE DE DOCTORAT
présentée par

Marc CHEVALLIER

Laboratoire d'Informatique de Paris Nord

pour l'obtention du grade de
DOCTEUR EN INFORMATIQUE

soutenue le 23/11/2022 devant le jury d'examen composé de :

Mohamed Quafafou,	Aix-Marseille Université	Rapporteur
Gilles Bernard,	Université Paris 8	Rapporteur
Younès Bennani,	Université Sorbonne Paris Nord	Président du jury
Nistor Grozavu,	Université Cergy-Pontoise	Examinateur
Nicoleta Rogovschi,	Université Paris Cité	Examinatrice
Faouzi Boufarès,	Université Sorbonne Paris Nord	Directeur de thèse
Charly Clairmont,	Société Synaltic	Invité

REMERCIEMENTS

Je souhaite remercier Marc Sallières, Charly Clairmont et Faouzi Boufarès sans qui cette aventure n'aurait jamais pu voir le jour. Un grand merci à Nistor et Nicoleta pour leurs conseils avisés. Je remercie ensuite les membres du jury, qui m'ont fait l'honneur d'étudier mon travail.

Je souhaite également remercier tous mes collègues du LIPN qui m'ont conseillé et épaulé quand j'en avais besoin. J'ai une pensée pour tous mes collaborateurs de chez Synaltic qui ont subi mes vociférations durant trois longues années. J'aimerais remercier particulièrement toutes les personnes qui m'ont aidé dans la relecture de mes travaux.

Merci à mes parents pour leur soutien durant toutes ces années. Un grand merci à MSDLR qui aura toujours su me réconforter durant les moments de doutes. J'ai aussi un remerciement spécial pour Astrid qui m'a supporté au quotidien avec une patience infinie durant toutes ces années de thèse.

Enfin j'ai une pensée empreinte d'émotion pour mon grand-père Jean qui nous a quittés durant la rédaction de ce manuscrit.

ABSTRACT

The digital transformation that has been rapidly happening within companies over the last few decades has led to a massive production of data. Once the problems related to the storage of those data have been solved, its use within Business Intelligence (BI) or Machine Learning (ML) has become a major objective for companies in order to make their data profitable. But the exploitation of the data is complex because it is not well documented and often contains many errors. It is in this context that the fields of data profiling and data quality (DQ) have become increasingly important. Profiling aims at extracting informative metadata from the data and data quality aims at quantifying the errors in the data. Profiling being a prerequisite to data quality, we have focused our work on this subject through the use of metadata vectors resulting from simple profiling actions. These simple information vectors have allowed us to perform advanced profiling tasks, in particular the prediction of complex semantic types using machine learning. The metadata vectors we used are large and are therefore affected by the curse of dimensionality. This term refers to a set of performance problems that occur in machine learning when the number of dimensions of the problem increases. One method to solve these problems is to use genetic algorithms to select a subset of dimensions with good properties. In this framework we have proposed improvements : on one hand, a non-random initialization of the individuals composing the initial population of the genetic algorithm, on the other hand, a modification to the genetic algorithm with aggressive mutations in order to improve its performance (GAAM).

Keywords : Data profiling, Machine learning, Feature selection, Genetic algorithm

RÉSUMÉ

La transformation digitale qui s'est effectuée de manière rapide aux cours des dernières décennies au sein des entreprises a donné lieu à une production massive de données. Une fois les problèmes liés au stockage de ces données résolus, leur utilisation au sein de la Business Intelligence (BI) ou du Machine Learning (ML) est devenue un objectif majeur des entreprises afin de rentabiliser leurs données. Mais l'exploitation de ces données s'avère complexe car elles sont très peu documentées et contiennent très souvent de nombreuses erreurs. C'est dans ce contexte que les domaines du profilage des données et de la qualité des données (QD) ont pris de plus en plus d'importance, le profilage ayant pour but d'extraire des méta-données informatives sur les données et la qualité des données de quantifier les erreurs dans les données. Le profilage étant un pré-requis à la qualité des données nous avons concentré nos travaux sur ce sujet au travers de l'utilisation de vecteurs de méta-données issu d'action de profilage simple. Ces vecteurs d'informations simples nous ont permis de réaliser des tâches de profilage avancées, en particulier la prédiction de type sémantique complexe au moyen d'algorithmes d'apprentissage artificiel. Les vecteurs de méta-données que nous avons utilisés sont de grande taille et sont donc affectés par la malédiction de la grande dimension. Ce terme regroupe un ensemble de problèmes de performance survenant en apprentissage artificiel quand le nombre de dimensions du problème augmente. Une méthode pour résoudre ces problèmes est d'utiliser des algorithmes génétiques pour sélectionner un sous-ensemble de dimensions ayant de bonnes propriétés. Dans ce cadre nous avons proposé des améliorations : d'une part, une initialisation non aléatoire des individus composant la population initiale de l'algorithme génétique, d'autre part, des modifications pour l'algorithme génétique avec des mutations agressives afin d'améliorer ses performances (GAAM).

Mots-Clés : Profilage des données, Apprentissage Artificiel, Sélection de caractéristiques, Algorithme génétique

TABLE DES MATIÈRES

Abstract **2**

Résumé **3**

1 Introduction Générale **19**
1.1 Introduction 19
1.2 Les données 21
1.3 Contexte 22
1.4 Problématique 23
1.5 Résumé des chapitres 24
1.5.1 État de l'art 24
1.5.2 Vecteurs de métadonnées pour le profilage des données 25
1.5.3 Sélection de caractéristiques par algorithmes génétiques 25

2 État de l'art **27**
2.1 Introduction 28
2.2 Profilage des données 28
2.2.1 Profilage des données mono-colonnes 28
2.2.1.1 Cardinalité 29
2.2.1.2 Distribution de valeurs 30
2.2.1.3 Types de données, schémas, domaines 31
2.2.1.4 Approximations 33
2.2.2 Profilage des données multi-colonnes 33
2.2.2.1 Les différents types de contraintes 33
2.2.3 Profilage des données Mono et Multi-lignes 37
2.2.4 Exemples d'outils de profilage 38
2.3 L'apprentissage artificiel (Machine Learning) 40
2.3.1 L'apprentissage automatique 40
2.3.1.1 L'apprentissage non-supervisé 40

2.3.1.2 L'apprentissage par renforcement 41
2.3.1.3 L'apprentissage Supervisé 41
2.3.2 Les modèles supervisés que nous avons utilisés 44
2.3.2.1 Classifieur bayésien naïf 44
2.3.2.2 Arbres décisionnels 45
2.3.2.3 Forêt d'arbres décisionnels 47
2.3.2.4 Boosting 49
2.3.2.5 Gradient tree Boosting 49
2.3.2.6 Régression logistique 50
2.3.2.7 Réseau de neurones 51
2.3.2.8 Stacking 52
2.3.2.9 Calibration 52
2.3.2.10 Plongement de mots 53
2.3.3 Les modèles non-supervisés que nous avons utilisés . . 53
2.3.3.1 Analyse en composante principale 53
2.3.3.2 Auto-Encodeur 54
2.3.3.3 Stochastic Neighbor Embedding 55
2.3.3.4 t-Distributed Stochastic Neighbor Embedding 56
2.4 Sélection de caractéristiques 57
2.4.1 Sélection séquentielle de caractéristiques 60
2.4.2 Algorithmes génétiques 60
2.5 Conclusion du chapitre 62

3 Vecteurs de métadonnées pour le profilage des données 63
3.1 Introduction 64
3.2 Création du vecteur de caractéristiques 64
3.3 La détection des types sémantiques 68
3.3.1 Introduction 68
3.3.2 Apprentissage pour la détection de types sémantiques . 69
3.3.3 Génération des données 70
3.3.4 Extractions des caractéristiques 71
3.3.5 Modèle d'apprentissage 71
3.3.6 Protocole expérimental 72
3.3.7 Conclusion 79
3.4 Détection des Colonnes quasi-dupliquées 79
3.4.1 Introduction 79
3.4.2 Description générale 80
3.4.3 Description des caractéristiques 81
3.4.4 Principe des algorithmes 81

3.4.5 Généralisation 83
3.4.6 Expériences 84
3.4.7 Résultats 91
3.4.8 Conclusion 92
3.5 La détection des jeux de données quasi-dupliquées 93
3.5.1 Introduction 93
3.5.2 Description du problème 93
3.5.3 Caractéristiques 93
3.5.4 Algorithmes 94
3.5.5 Expériences 97
3.5.5.1 Influence du paramètre ζ 97
3.5.5.2 Influence du nombre de colonnes 97
3.5.5.3 Évaluation de plusieurs classifieurs 97
3.5.5.4 Optimisations 98
3.5.6 Résultats 99
3.5.6.1 Influence du paramètre ζ 99
3.5.6.2 Influence du nombre de colonnes 99
3.5.6.3 Évaluation de plusieurs classifieurs 99
3.5.6.4 Optimisations 105
3.5.7 Conclusion 106
3.6 Conclusion du chapitre 107

4 Sélection de Caractéristiques par algorithme génétique 109
4.1 Introduction 109
4.2 GAAM 112
4.3 Ensemencement de population initiale 114
4.3.1 Éclectique GA (EGA) 114
4.3.2 Expériences 115
4.3.2.1 Conditions d'expérience 115
4.3.2.2 Techniques de création de la population initiale 116
4.3.3 Résultats 118
4.3.3.1 Résultats avec l'algorithme EGA 118
4.3.3.2 Résultats avec l'algorithme GAAM 119
4.3.3.3 Comparaison avec des méthodes de réduction dimensionnelle 122
4.3.4 Conclusion 122
4.4 Échange entre la taille de la population et le taux de mutation pour GAAM 123
4.4.1 Introduction 123

4.4.2 mGAAM 123
4.4.3 Expériences 124
4.4.4 Résultats 126
4.4.5 Test sur les vecteurs de meta-données 128
4.4.6 Conclusion 133
4.5 Conclusion du chapitre 134

5 Conclusion **135**
5.1 Contributions 135
5.2 Futures directions 137

References **139**

Annexes **155**

LISTE DES FIGURES

1.1 Volume de données/informations créées, copiées et consommées de 2010 à 2025 [Sta21] 20
1.2 Composantes classiques de la qualité des données 21
1.3 Exemple de source de données au format CSV, (Les données sont extraites d'une liste d'étudiants) 21
1.4 Définition d'une source de données DS 22
1.5 Les partenaires de Synaltic au fil du temps 23
1.6 Les clients de Synaltic 23

2.1 Aperçu des métadonnées et de leurs utilisations [Har20]. . . . 28
2.2 Les profilages des données mono-colonnes 29
2.3 Un exemple d'histogramme sur kaggle 30
2.4 Treillis de recherche de DF [AGNP18] 36
2.5 Exemple d'usage du profilage des données [AGNP18] 38
2.6 Exemple du profilage de données avec TRIFACTA 39
2.7 Exemple de résultat proposé par Talend Data Quality 39
2.8 Schématisation du domaine de l'intelligence artificielle 40
2.9 Représentation de l'apprentissage supervisé 41
2.10 Les types de modèles que nous avons utilisés 42
2.11 Matrice de confusion binaire 43
2.12 Validation croisée 4 plis 44
2.13 Arbre de décisions 46
2.14 Exemple d'arbre de décisions 47
2.15 Forêt d'arbres décisionnels 48
2.16 Structure d'un réseau de neurones totalement connectés (les lignes représentent les poids) pour la classification binaire . . . 51
2.17 Stacking de classifieurs 52
2.18 Auto-encodeur [Bur19] 54
2.19 Réduction en deux dimensions ACP 58

2.20 Réduction en deux dimensions auto-encodeur 58
2.21 Réduction en deux dimensions t-SNE 58
2.22 La sélection de caractéristiques 59
2.23 Fonctionnement d'un algorithme génétique pour la sélection de caractéristiques 61

3.1 Processus de création du vecteur de caractéristiques 65
3.2 Moyenne de la norme euclidienne du vecteur caractéristique utilisant la méthode de Sherlock pour plusieurs longueurs de vecteur (pour une colonne de type sémantique FILM-FRENCH) 66
3.3 Moyenne de la norme euclidienne de notre vecteur de caractéristiques pour plusieurs tailles de vecteur (pour une colonne de type sémantique FILM-FRENCH) 67
3.4 Temps d'exécution en fonction du nombre de lignes 68
3.5 Schéma récapitulatif de la méthode proposée 69
3.6 Taux de bonne reconnaissance sur les jeux de données de test de la version "haute" pour différentes valeurs de ω. 74
3.7 Taux de bonne reconnaissance sur les jeux de données de test de la version "basse" pour différentes valeurs de ω. 75
3.8 Taux de bonne reconnaissance des classes connues et inconnues pour différents seuils avec le classifieur RF 'haute'. 77
3.9 Taux de bonne reconnaissance des classes connues et inconnues pour différents seuils avec le classifieur Catboost 'haute' 77
3.10 Projection en deux dimensions de NDC et NNDC, pour des colonnes d'adresses emails avec $\alpha = 1$ 85
3.11 Projection en deux dimensions de NDC et NNDC, pour des colonnes de noms avec $\alpha = 2$ 85
3.12 Taux de bonne reconnaissance pour différentes valeurs de α sur des colonnes d'adresses emails 86
3.13 Taux de bonne reconnaissance pour différentes valeurs de α sur des colonnes de noms 87
3.14 Concaténation des colonnes avant extraction des caractéristiques 94
3.15 NNDDS et NDDS pour $\alpha \in [0, 3]$ et $\zeta \in [0, 3]$, 10 colonnes avec la formule 3.6 96
3.16 Taux de bonne reconnaissance, RF classifieur pour $\alpha \in [0, 3]$ et $\zeta \in [0, 3]$ avec la formule 3.6 102
3.17 Taux de bonne reconnaissance, RF classifieur pour $\alpha \in [0, 3]$ et $\zeta \in [0, 3]$ avec la formule 3.7 102

3.18 Taux de bonne reconnaissance, Catboost classifieur pour $\alpha \in [0,3]$ et $\zeta \in [0,3]$ avec la formule 3.6 102
3.19 Taux de bonne reconnaissance, Catboost classifieur pour $\alpha \in [0,3]$ et $\zeta \in [0,3]$ avec la formule 3.7 102
3.20 Taux de bonne reconnaissance, LGBM classifieur pour $\alpha \in [0,3]$ and $\zeta \in [0,3]$ avec la formule 3.6 103
3.21 Taux de bonne reconnaissance, LGBM classifieur pour $\alpha \in [0,3]$ and $\zeta \in [0,3]$ avec la formule 3.7 103
3.22 Taux de bonne reconnaissance, TabNet classifieur pour $\alpha \in [0,3]$ and $\zeta \in [0,3]$ avec la formule 3.6 103
3.23 Taux de bonne reconnaissance, TabNet classifieur pour $\alpha \in [0,3]$ et $\zeta \in [0,3]$ avec la formule 3.7 103
3.24 Taux de bonne reconnaissance, Adaboost classifieur pour $\alpha \in [0,3]$ et $\zeta \in [0,3]$ avec la formule 3.6 103
3.25 Taux de bonne reconnaissance, Adaboost classifieur pour $\alpha \in [0,3]$ et $\zeta \in [0,3]$ avec la formule 3.7 103
3.26 Taux de bonne reconnaissance, Stack classifieur pour $\alpha \in [0,3]$ et $\zeta \in [0,3]$ avec la formule 3.6 104
3.27 Taux de bonne reconnaissance, Stack classifieur pour $\alpha \in [0,3]$ et $\zeta \in [0,3]$ avec la formule 3.7 104

4.1 Illustration du fonctionnement de l'algorithme GAAM 114
4.2 Boite à moustache du taux de reconnaissance pour 1000 individus générés avec chaque méthode 117
4.3 Évolution du taux de bonne reconnaissance moyen calculé sur 10 simulations en utilisant SSM sur Semeion 118
4.4 Évolution du taux de bonne reconnaissance moyen calculé sur 10 simulations en utilisant SSM sur Madelon 118
4.5 Évolution du taux de bonne reconnaissance moyen calculé sur 10 simulations en utilisant ESM sur Semeion 119
4.6 Évolution du taux de bonne reconnaissance moyen calculé sur 10 simulations en utilisant ESM sur Madelon 119
4.7 Évolution du taux de bonne reconnaissance moyen calculé sur 10 simulations en utilisant SSM sur Semeion 120
4.8 Évolution du taux de bonne reconnaissance moyen calculé sur 10 simulations en utilisant SSM sur Madelon 120
4.9 Évolution du taux de bonne reconnaissance moyen calculé sur 10 simulations en utilisant ESM sur Semeion 120

4.10 Évolution du taux de bonne reconnaissance moyen calculé sur 10 simulations en utilisant ESM sur Madelon 120
4.11 Taux de bonne reconnaissance moyen sur Leaf pour plusieurs tailles de population 127
4.12 Taux de bonne reconnaissance moyen sur Micro-Mass pour plusieurs tailles de population 127

LISTE DES TABLEAUX

2.1 Exemple de données client 34
2.2 Exemple de données Signe astrologique 36
2.3 Exemple de profilage de ligne 37
2.4 Exemple de jeu de données pour les arbres de décisions 45

3.1 Écart-type de la norme euclidienne du vecteur caractéristique de Sherlock pour plusieurs tailles de vecteur (pour une colonne de type sémantique FILM-FRENCH) pour les deux méthodes 67
3.2 Temps d'exécution en fonction du nombre de lignes 68
3.3 Taux de bonne reconnaissance des modèles 73
3.4 F1-score 'macro' des modèles 74
3.5 Les 10 meilleures caractéristiques pour Catboost, classées par ordre décroissant par leur score d'impureté de Gini. 75
3.6 Les 10 meilleures caractéristiques pour RF, classées par ordre décroissant par leur score d'impureté de Gini. 76
3.7 Taux de bonne reconnaissance des classes connues et inconnues pour différents seuils avec les classifieurs Catboost RF et Catboost classifier part 1. 78
3.8 Taux de bonne reconnaissance des classes connues et inconnues pour différents seuils avec les classifieurs RF et Catboost classifier part 2. 78
3.9 Taux de bonne reconnaissance pour la première expérience avec les colonnes de mails et de noms part 1. 86
3.10 Taux de bonne reconnaissance pour la première expérience avec les colonnes de mails et de noms part 2. 87
3.11 Taux de bonne reconaissance pour la deuxième expérience sur U pour 2 classifieurs 88
3.12 Taux de bonne reconnaissance sur U' pour 3 classifieurs 88
3.13 Précision sur U' pour 3 classifieurs 88

3.14 Rappel sur U' pour 3 classifieurs 89
3.15 F1-score sur U' pour 3 classifieurs 89
3.16 Taux de bonne reconnaissance pour 2 modèles entraînés sur plusieurs alpha pour la dernière expérience 89
3.17 Rappel pour 2 modèles entraînés sur plusieurs alpha pour la dernière expérience 89
3.18 Précision pour 2 modèles entraînés sur plusieurs alpha pour la dernière expérience 90
3.19 F1-score pour 2 modèles entraînés sur plusieurs alpha pour la dernière expérience 90
3.20 Taux de bonne reconnaissance pour plusieurs valeurs de ζ . . . 99
3.21 Taux de bonne reconnaissance en fonction du nombre de colonnes utilisées pour construire le jeu de données d'entraînement et le jeu de données de test. 100
3.22 Taux de bonne reconnaissance de six classifieurs pour différents couples de (α,ζ) avec la formule 3.6 part 1. 101
3.23 Taux de bonne reconnaissance de six classifieurs pour différents couples de (α,ζ) avec la formule 3.6 part 2. 101
3.24 Taux de bonne reconnaissance de six classifieurs pour différents couples de (α,ζ) avec la formule 3.7 part 1. 101
3.25 Taux de bonne reconnaissance de six classifieurs pour différents couples de (α,ζ) avec la formule 3.7 part 2. 102
3.26 Top 10 des caractéristiques pour les modèles Catboost et RF avec la formule 3.6, classé par rang descendant. 104
3.27 Top 10 des caractéristiques pour les modèles Catboost et RF avec la formule 3.7, classé par rang descendant. 105
3.28 Taux de bonne reconnaissance pour 6 classifieurs en utilisant les deux méthodes et les deux formules 106

4.1 Description des jeux de données utilisiés 110
4.2 Taux de bonne reconnaissance pour 10 caractéristiques, (validation croisée 3 plis) pour SFS et taux de bonne reconnaissance médian (50 simulations) pour GAAM part a 110
4.3 Taux de bonne reconnaissance pour 10 caractéristiques, (validation croisée 3 plis) pour SFS et taux de bonne reconnaissance médian (50 simulations) pour GAAM part b 111
4.4 Taux de bonne reconnaissance moyen pour 10 caractéristiques, obtenu en répétant 50 fois une séparation 75% apprentissage 25% test part a 111

4.5 Taux de bonne reconnaissance moyen pour 10 caractéristiques, obtenu en répétant 50 fois une séparation 75% apprentissage 25% test part b 112
4.6 Description des jeux de données testées pour l'ensemencement d'AG 116
4.7 Taux de bonne reconnaissance (Tbr) pour le jeu de données Madelon, 10 simulations 121
4.8 Taux de bonne reconnaissance (Tbr) pour le jeu de données Semeion, 10 simulations 121
4.9 Taux de bonne reconnaissance moyen pour EGA et GAAM 20 caractéristiques, pour les deux techniques d'ensemencement, obtenu en répétant 50 fois une séparation 75% apprentissage 25% 121
4.10 Résultats avec 20 caractéristiques 122
4.11 Résultats sur la moitié du nombre de caractéristiques 122
4.12 Résultat sur Micro-mass en utilisant l'algorithme 14 pour calculer le taux de mutation 128
4.13 Résultat de l'algorithme mGAAM sur plusieurs jeux de données comparativement à GAAM utilisant 10 individus, 10 caractéristiques et 100 itérations, les résultats sont obtenus sur 50 simulations part a1. 129
4.14 Résultat de l'algorithme mGAAM sur plusieurs jeux de données comparativement à GAAM utilisant 10 individus, 10 caractéristiques et 100 iterations, les résultats sont obtenus sur 50 simulations part a2. 130
4.15 Résultat de l'algorithme mGAAM sur plusieurs jeux de données comparativement à GAAM utilisant 10 individus, 10 caractéristiques et 100 iterations, les résultats sont obtenus sur 50 simulations part b1. 131
4.16 Résultat de l'algorithme mGAAM sur plusieurs jeux de données comparativement à GAAM utilisant 10 individus, 10 caractéristiques et 100 iterations, les résultats sont obtenus sur 50 simulations part b2. 132
4.17 Taux de bonne reconnaissance moyen pour GAAM et mGAAM (60 inds, formule 14), 10 caractéristiques, obtenu en répétant 50 fois une séparation 75% apprentissage 25% test part part a. 133
4.18 Taux de bonne reconnaissance moyen pour GAAM et mGAAM (60 inds, formule 14, 10 caractéristiques, obtenu en répétant 50 fois une séparation 75% apprentissage 25% test part b. . . . 133

A.1 Liste des 57 types sémantiques inclus dans cette étude part a. 155
A.2 Liste des 57 types sémantiques inclus dans cette étude part b. 156

ABRÉVIATIONS

Symbole	**Signification**
ACP	Analyse en composantes principales
AG	Algorithme génétique
BI	Informatique décisionnelle
CART	Classification And Regression Trees
CSV	Valeurs séparées par des virgules
Catb	Catboost
DC	Contraintes de déni
DF	Dépendance fonctionnelle
DHC	Taux de croisement haut décroissant
DS	Source de données
EGA	Algorithme génétique éclectique
ESM	Méthode d'ensemencement élitiste
FN	Faux Négatif
FP	Faux Positif
GAAM	Algorithme génétique avec des mutations agressives
GLOVE	Global Vectors for Word Representation
ILM	Taux de mutation bas croissant
LGBM	Light Gradient Boosting Machine
ML	Apprentissage automatique
NDC	Colonne quasi dupliquée
NDDS	Ensemble de données quasi dupliqué
NN	Réseau de neurones artificiels
NNDC	Colonne non dupliquée
NNDDS	Ensemble de données non dupliqué
PAV	Pair-adjacent violators
PLI	Liste d'indice de position
QD	Qualité des données
RF	Forêt d'arbres décisionnels
Sato	Semantic Type detection with table context
SFS	Sequential Feature Selection
SSM	Méthode d'ensemencement standard
Tbr	Taux de bonne reconnaissance
t-SNE	t-Stochastic Neighbor Embedding
VN	Vrai Négatif
VP	Vrai Positif

CHAPITRE 1

INTRODUCTION GÉNÉRALE

1.1 Introduction 19
1.2 Les données 21
1.3 Contexte 22
1.4 Problématique 23
1.5 Résumé des chapitres 24
1.5.1 État de l'art 24
1.5.2 Vecteurs de métadonnées pour le profilage des données 25
1.5.3 Sélection de caractéristiques par algorithmes génétiques 25

1.1 Introduction

La profilage des données est un sujet qui a émergé en même temps que l'apparition des premières bases de données. Le profilage des données que l'ont peut définir comme "l'ensemble des activités et des processus conçus pour déterminer les métadonnées d'un ensemble de données donné" [AGNP18] est étroitement lié au sujet de la qualité des données, "La qualité des données concerne la préservation du sens des données tel qu'il est perçu par les concepteurs et les utilisateurs d'une application de base de données" [Bro80]. Les deux sujets largement présents dans la littérature depuis les années 1980 ont pris de l'ampleur avec la révolution des technologies de l'information et de la communication. En effet l'augmentation des sources de données et des capacités matérielles ont permis une production de données toujours plus grande.

Ce phénomène s'est traduit dans les entreprises par une accumulation de données, souvent de manière anarchique. Les données ont été stockées rapidement sans que leurs usages futurs ne soient réellement déterminés. De ce fait les données ont généralement été sauvegardées avec très peu de méta-informations pour les caractériser. En l'absence de celles-ci le ré-usage de ces données est difficile sans passer par une étape de profilage.

Avec l'avènement de la Business Intelligence (BI) et du Machine Learning (ML) les entreprises sont de plus en plus guidées par les données. D'importantes décisions sont prises à partir de celles-ci et les anomalies dans les

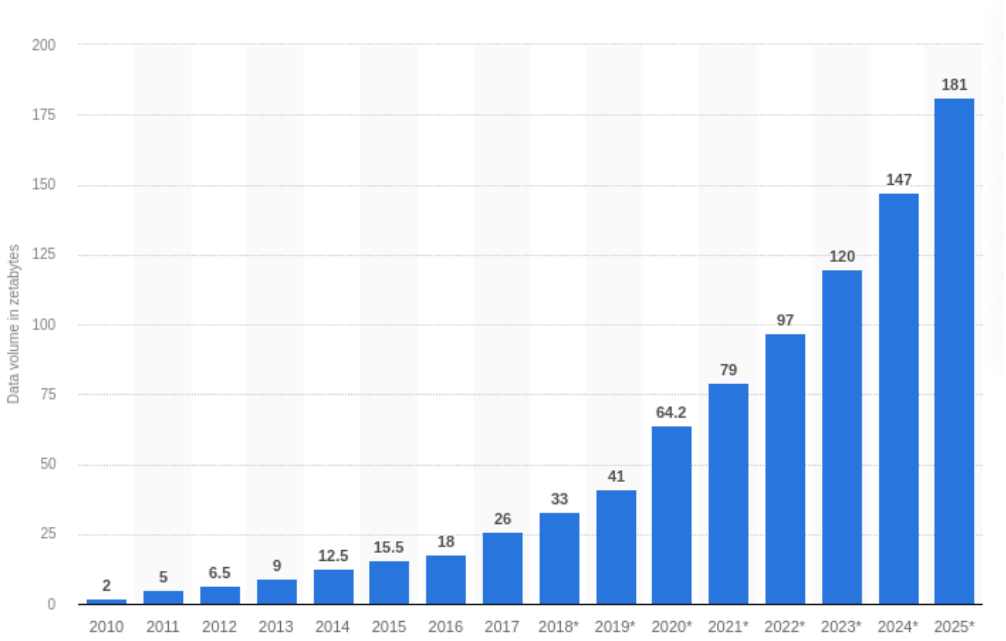


FIGURE 1.1 – Volume de données/informations créées, copiées et consommées de 2010 à 2025 [Sta21]

données (manque de qualité) ont un impact très important pour les entreprises [Red98, Red16, BFI+22].

Il n'existe aucune mesure ou métrique réellement universelle pour quantifier la qualité des données, car la qualité des données est fondamentalement contextuelle. Chaque acteur du monde de la donnée a sa propre vision de la qualité. Cependant pour orienter le profilage, il faut définir des métriques de base communes permettant chacune d'évaluer un aspect des anomalies que l'on peut rencontrer dans les données.
Finalement c'est l'usage que l'on va avoir de la donnée qui va déterminer ce que l'on juge comme rédhibitoire ou non. La figure 1.2 présente une vision classique des composantes les plus importantes qui définissent la qualité des données. Nous allons maintenant les énumérer.

La complétude caractérise la présence ou l'absence de toutes les données nécessaires à l'usage de ces dernières. La temporalité caractérise le fait que les données soient périmées ou non et que les données les plus récentes soient disponibles en temps voulu. L'intégrité caractérise le respect des relations existantes entre les données ainsi que leurs sens. La consistance caractérise la présence ou non d'aberration dans les données telles que les données dupliquées. La précision caractérise la véracité des données et plus précisément le fait que ces données viennent d'une source fiable ou non, mais aussi si celles-ci décrivent bien l'entité qu'elles représentent. La validité des données représente le fait que les données respectent ou non les règles établies pour

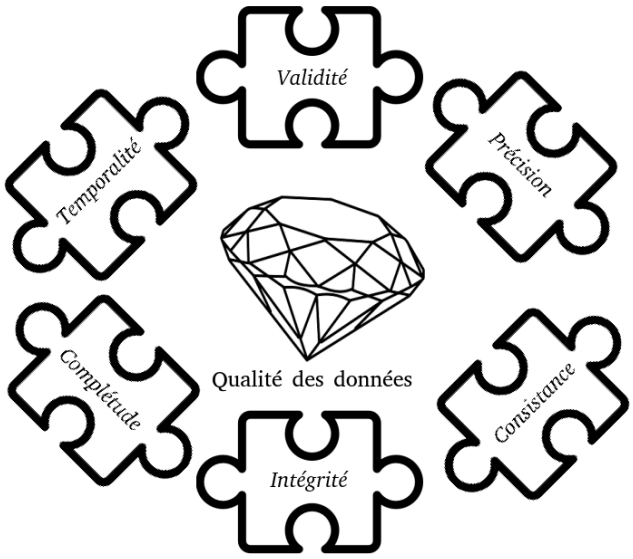


FIGURE 1.2 – Composantes classiques de la qualité des données

leur utilisation (par exemple, si les plages de valeurs prises sont dans les plages de valeurs attendues pour l'utilisation ou non). Ainsi le profilage des données peut avoir pour but d'extraire des metadonnées permettant de mesurer, d'évaluer ou d'améliorer ces points.

1.2 Les données

Dans ce document on s'intéressera aux données de type tabulaires stockées par exemple dans une architecture de type lac de données. Ces données pouvant provenir de tables dans des bases de données relationnelles ou de fichiers de type CSV [Sha05].

IG001;19/06/2021 14:28;BERHILI;FAÏZA;Femme;M2EID2;faizaberhili@gmail.com;;IG;Villetaneuse;France;1993-octobre-20;28;44.792718, -0.608576
IG002;19/06/2021 14:28;BOUSSAC;EMILIANO;Homme;M2EID2;emiliano.boussac@gmail.com;;IG;Villetaneuse;France;1996-janvier-11;25;44.79622, -0.600218
IG003;19/06/2021 14:28;ALSADIK;AMMAR;Homme;M2EID2;ammar.alsadik@edu.univ-paris13.fr;;IG;Villetaneuse;France;1998-juin-19;23;43.6189, 3.8558
IG004;19/06/2021 14:28;CHAMBRIER;JULIAN;Homme;M2EID2;julianchambrier@hotmail.fr;;IG;Villetaneuse;France;1998-août-04;23;48.1146, -1.6568
IG005;19/06/2021 14:28;MALKI;SALMA;Femme;M2EID2;salmamalki5@gmail.com;;IG;Villetaneuse;France;1995-septembre-06;26;48.113, -1.70415
IG006;19/06/2021 14:28;SABOUNI;SOUKAYNA;Femme;M2EID2;sabouni.soukayna@gmail.com;;IG;Villetaneuse;France;1994-mai-22;27;45.1913, 5.71753
IG007;19/06/2021 14:28;CHFADI;SARA;Femme;M2EID2;chfadi.sara@gmail.com;;IG;Villetaneuse;France;1998-avril-18;23;48.6962, 6.17634
IG008;19/06/2021 14:28;EL HAIL;MOHAMED;Homme;M2EID2;mohamedelhail21@gmail.com;;IG;Villetaneuse;France;1997-février-09;24;50.636459, 3.074534
IG009;19/06/2021 14:28;BABORI;YASMINE;Femme;M2EID2;yasminebabori@hotmail.com;;IG;Villetaneuse;France;1994-août-08;27;50.634181, 3.048712
IG010;19/06/2021 14:28;ESSEBBABI;NOUR;Femme;M2EID2;essebbabinr@gmail.com;;IG;Villetaneuse;France;1998-janvier-06;23;50.6063, 3.13643
IG011;19/06/2021 14:28;NDIAYE;MOHAMED;Homme;M2EID2;amethndiayes17@gmail.com;;IG;Villetaneuse;France;1993-novembre-28;28;50.635423, 3.045179
IG012;19/06/2021 14:28;SENE;AÏSSATOU BÂ;Femme;M2EID2;aissatoubasene97@gmail.com;;IG;Villetaneuse;France;1998-avril-23;23;50.6314, 3.07497
IG013;19/06/2021 14:28;MBENGUE;IBRAHIMA;Femme;M2EID2;imbengue027@gmail.com;;IG;Villetaneuse;France;1993-décembre-10;28;45.770548, 3.087722
IG014;19/06/2021 14:28;TCHEDRE;NAPO TIYADJA;Femme;M2EID2;tchedrace@outlook.com;;IG;Villetaneuse;France;1995-mai-03;26;43.446174, -1.553892
IG015;19/06/2021 14:28;DIALLO;MAMADOU KORKA;Femme;M2EID2;mamadoukorka.diallo@edu.univ-paris13.fr;;IG;Villetaneuse;France;1996-janvier-06;25;48.58582, 7.73711
IG016;19/06/2021 14:28;SENE;MAME ASTOU;Femme;M2EID2;senemameastou@gmail.com;;IG;Villetaneuse;France;1997-novembre-21;24;48.58069, 7.73739
IG017;19/06/2021 14:28;DJENADI;SABRINA;Femme;M2EID2;sabrina.djenadi@edu.univ-paris13.fr;;IG;Villetaneuse;France;1995-novembre-07;26;47.7323, 7.31485
IG018;19/06/2021 14:28;MEDJDOUBI;LYNDA;Femme;M2EID2;lynda.medjdoubi@edu.univ-paris13.fr;;IG;Villetaneuse;France;1998-décembre-12;23;45.747, 4.83577
IG019;19/06/2021 14:28;BIAN;YIPING;Femme;M2EID2;bianyiping@gmail.com;;IG;Villetaneuse;France;1997-juin-19;24;45.784, 4.77027
IG020;19/06/2021 14:28;QIAN;XIAOTONG;Femme;M2EID2;xiaotong.qian@edu.univ-paris13.fr;;IG;Villetaneuse;France;1997-octobre-23;24;45.7535, 4.81245

FIGURE 1.3 – Exemple de source de données au format CSV, (Les données sont extraites d'une liste d'étudiants)

Soit $C = \{Colonne_1, Colonne_2, \ldots, Colonne_n\}$ un ensemble de noms de colonnes. Chaque colonne est définie sur un domaine. Une source de données (Data source DS), définie sur les colonnes $Colonne_1, Colonne_2, \ldots Colonne_n$

est un sous-ensemble du produit cartésien des domaines de définition de chacune des colonnes. Chaque sous-ensemble constitue une instance (c'est-à-dire une occurrence) de la source de données. Il s'agit donc d'un ensemble de n-uplets. Chaque n-uplet est un élément du sous-ensemble du produit cartésien des n domaines. On notera : t un n-uplet, t[C_i] une valeur v de la colonne C_i dans t.

L'exemple donné dans la figure 1.3 constitue une occurrence de la source de données définie sur une vingtaine de colonnes (n=20), elle contient m lignes.

Un schéma est une définition de la structure complète d'une source de données. Il liste toutes les colonnes sur lesquelles porte la source de données. Un schéma décrit toutes les propriétés d'une source, à savoir : (i) les noms des colonnes, (ii) les aspects syntaxiques de chacune (c'est à dire les domaines syntaxiques ou les types de données), (iii) les aspects sémantiques des colonnes (les contraintes) (iv) des commentaires éventuels rattachés aux différentes colonnes.

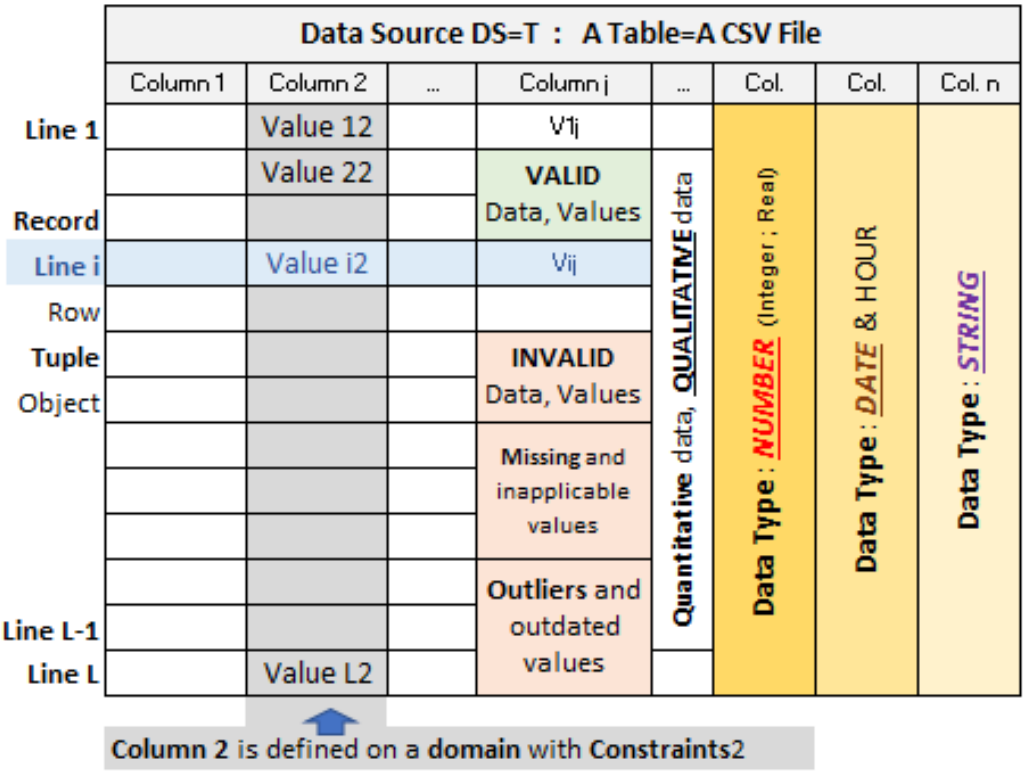


FIGURE 1.4 – Définition d'une source de données DS

1.3 Contexte

Cette thèse est une thèse Cifre qui s'est déroulée en partenariat entre le Laboratoire d'Informatique Paris Nord (LIPN) et l'entreprise Synaltic.

Synaltic est une entreprise de services numériques spécialisée dans le monde de la donnée. Ses locaux sont situés à Vincennes, elle a été fondée en 2004 par Charly Clairmont et Marc Salière. Synaltic compte aujourd'hui 30 collaborateurs et réalise un chiffre d'affaires de 2 941 211 euros (2021).

L'entreprise a toujours voulu être pionnière, orientée vers le logiciel libre et l'innovation.
Il est capital pour Synaltic d'identifier les technologies émergentes qui dans le futur seront leader dans leur secteur de marché. Savoir déceler et maîtriser ces nouveaux outils permet à Synaltic de devenir un partenaire de la première heure des outils de demain.

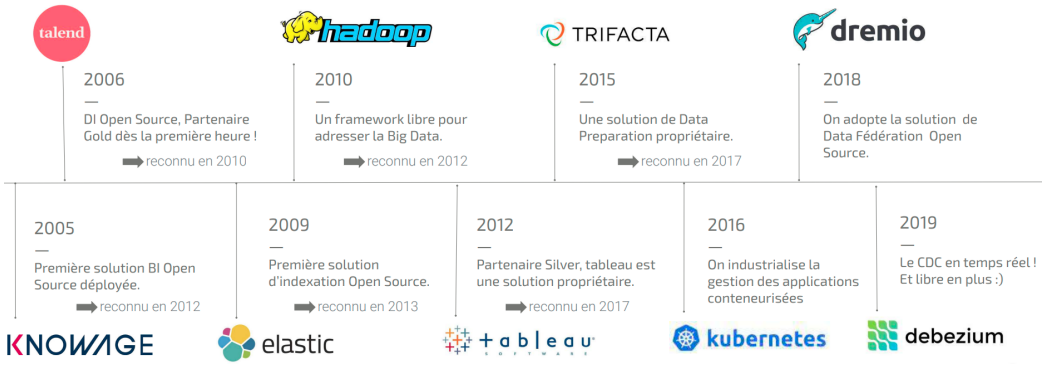


FIGURE 1.5 – Les partenaires de Synaltic au fil du temps

Cette capacité à identifier les innovations porteuses et les adapter aux besoins des clients a permis à Synaltic d'accompagner 160 clients ces dernières années.

FIGURE 1.6 – Les clients de Synaltic

C'est ainsi que dans un désir d'explorer les possibilités de diversifications de l'entreprise et de participer activement à l'innovation que Synaltic a financé cette thèse Cifre.

1.4 Problématique

Dans ce document on s'intéresse à l'analyse et l'extraction de métadonnées sur les données de structure tabulaire. Quelle que soit la source de ces données (une table dans une base de données relationnelles, un lac de données

ou un simple fichier CSV), il est très fréquent qu'il manque des informations cruciales venant décrire les données. Ce manque d'information rend difficile l'exploration, l'intégration, l'analyse des données ou encore l'évaluation de la qualité de celles-ci. Ces problèmes ont ainsi créé le besoin d'extraire automatiquement des méta-informations sur les données afin de mieux les décrire.
Nous nous sommes intéressés plus particulièrement à déterminer par apprentissage automatique le domaine sémantique des données, celui-ci étant une information très souvent manquante dans les données. Lors de nos recherches sur ce sujet il nous est apparu que les méta-informations utilisées pour la découverte des types sémantiques pouvaient être utilisées pour d'autres sujets. Nous nous sommes ainsi intéressés à la détection de colonnes et de jeux de données doubles ou quasi-double (parmi un ensemble de jeux de données).

En effet, Synaltic explorant la possibilité d'héberger une plateforme de données open source, il est apparu opportun d'identifier de trop grandes similitudes entre des jeux de données. Enfin l'exploration des deux problématiques précédentes a fait naître le besoin d'explorer une troisième problématique. Celle la sélections de caractéristiques pour la classification.

1.5 Résumé des chapitres

Ce document s'articule autour de 3 chapitres principaux. Le premier donnant un aperçu du domaine du profilage des données et détaillant l'ensemble des algorithmes d'apprentissage automatique qui seront utilisés dans les chapitre suivant. Le second introduit le processus de création d'un vecteur de métadonnées qui sera ensuite utilisé pour identifier des types sémantiques, détecter des colonnes quasi-dupliquées et identifier des jeux de données quasi-dupliqués. Le dernier chapitre traite de la sélection de caractéristique à l'aide d'algorithmes génétique.

1.5.1 État de l'art

Ce premier chapitre, très général, aborde les deux thèmes principaux de cette thèse : le profilage des données et les modèles d'apprentissage automatique. Tout d'abord nous décrirons les principaux aspects du profilage monocolonne au travers des notions de cardinalités, distributions de valeurs et type de données. Nous nous intéresserons tout particulièrement à l'identification de type sémantique. Nous présenteront ensuite un aperçu des techniques de profilage multi-colonnes et multi-lignes. Nous présenterons aussi différents usages du profilage de données ainsi que des outils de profilages.
Nous présenterons ensuite les principaux paradigmes de l'apprentissage automatique, nous nous attarderons sur la description des modèles d'apprentissages supervisés et non-supervisés les plus utilisés dans ce document. Enfin

nous présenterons le principe de la sélection de caractéristiques ainsi que méthodes classiques pour effectuer cette tâche.

1.5.2 Vecteurs de métadonnées pour le profilage des données

Dans ce chapitre nous présenterons nos travaux en matière de profilages de données. Notre but est de générer des métadonnées complexes exploitables et stockable dans un data-catalogue à partir de métadonnées plus simples issues du profilage.

Tout d'abord nous introduirons une technique pour résumer une colonne en un vecteur de métadonnées. Cette méthode dérive de celle utilisée dans l'état de l'art pour d'identifier le type sémantique d'une colonne à l'aide d'apprentissage automatique.
Afin de lever certains verrous de l'état de l'art nous proposons une version modifiée de ces vecteurs mais aussi une manière de générer des données artificiellement. Nous introduisons ensuite une méthode utilisant de l'apprentissage automatique pour détecter 57 types sémantiques personnalisés, de plus nous présentons une méthode permettant de rejeter les types sémantiques inconnues. Enfin nous testons cette méthode sur les données de l'entreprise Synaltic et présentons les résultats en annexe.
Dans la deuxième section de ce chapitre, nous introduisons le concept de colonne quasi-dupliquée. Nous proposons ensuite une méthode pour générer de telles colonnes artificiellement. Nous proposons une méthode utilisant de l'apprentissage automatique pour discerner ce type de colonnes. Nous cherchons ensuite les meilleurs paramètres de génération utilisés pour maximiser les résultats. Ensuite dans la dernière partie de ce chapitre, nous introduisons le concept de jeu de données quasi-dupliqué. Enfin nous proposons à une méthode analogue à celle utilisée pour les colonnes quasi-dupliquées pour identifier les jeux de données quasi-dupliqués.

1.5.3 Sélection de caractéristiques par algorithmes génétiques

Ce chapitre est dédié à la sélection de caractéristiques pour l'apprentissage automatique. Celui-ci commence par une comparaison entre une méthode déterministe et un algorithme génétique sur cette tâche.
Le chapitre se poursuit sur la présentation d'un algorithme génétique conçu spécialement pour la sélection de caractéristiques. Cet algorithme nommé algorithme génétique avec des mutations agressives (GAAM) est au coeur de plusieurs expériences durant ce chapitre.
Dans la première section du chapitre nous nous intéresserons à l'ensemencement de population initiale d'algorithmes génétiques. C'est-à-dire à l'in-

jection d'individus non aléatoires dans la population initiale de l'algorithme. Cette stratégie s'étant montrée efficace pour d'autre type de problèmes. Nous présentons ainsi deux méthodes d'ensemencement de la population initiale à partir de solutions obtenues à l'aide d'une forêt d'arbre aléatoires. Des tests sont ensuite menés afin d'évaluer nos méthodes sur deux algorithmes génétiques.
Dans la suite du chapitre nous proposons une version modifiée de l'algorithme GAAM afin d'en améliorer les performances. L'amélioration réside dans l'usage d'un plus grande nombre d'individus dans la population au prix d'une réduction du taux de mutation. Notre version modifiée de l'algorithme est ensuite évaluée sur 17 jeux de données contre l'algorithme original.
Enfin, nous proposons une application de notre algorithme sur notre jeux de données utilisé pour la reconnaissance des types sémantiques.

CHAPITRE 2

ÉTAT DE L'ART

2.1 Introduction 28
2.2 Profilage des données 28
2.2.1 Profilage des données mono-colonnes 28
2.2.1.1 Cardinalité 29
2.2.1.2 Distribution de valeurs 30
2.2.1.3 Types de données, schémas, domaines 31
2.2.1.4 Approximations 33
2.2.2 Profilage des données multi-colonnes 33
2.2.2.1 Les différents types de contraintes 33
2.2.3 Profilage des données Mono et Multi-lignes 37
2.2.4 Exemples d'outils de profilage 38
2.3 L'apprentissage artificiel (Machine Learning) 40
2.3.1 L'apprentissage automatique 40
2.3.1.1 L'apprentissage non-supervisé 40
2.3.1.2 L'apprentissage par renforcement 41
2.3.1.3 L'apprentissage Supervisé 41
2.3.2 Les modèles supervisés que nous avons utilisés 44
2.3.2.1 Classifieur bayésien naïf 44
2.3.2.2 Arbres décisionnels 45
2.3.2.3 Forêt d'arbres décisionnels 47
2.3.2.4 Boosting 49
2.3.2.5 Gradient tree Boosting 49
2.3.2.6 Régression logistique 50
2.3.2.7 Réseau de neurones 51
2.3.2.8 Stacking 52
2.3.2.9 Calibration 52
2.3.2.10 Plongement de mots 53
2.3.3 Les modèles non-supervisés que nous avons utilisés . . 53
2.3.3.1 Analyse en composante principale 53
2.3.3.2 Auto-Encodeur 54
2.3.3.3 Stochastic Neighbor Embedding 55
2.3.3.4 t-Distributed Stochastic Neighbor Embedding 56
2.4 Sélection de caractéristiques 57
2.4.1 Sélection séquentielle de caractéristiques 60
2.4.2 Algorithmes génétiques 60
2.5 Conclusion du chapitre 62

2.1 Introduction

Ce chapitre se divise en deux grandes parties, l'une faisant un tour d'horizon du domaine du profilage des données et s'attardant spécifiquement sur les problématiques étudiées dans cette thèse. La seconde offre une vue globale de l'apprentissage artificiel et détailler les algorithmes utilisés dans les autres chapitres.

2.2 Profilage des données

Le profilage des données est un domaine consistant à extraire depuis les données des informations utiles. Ces dernières résument certains aspects des données cibles. Le but est informatif mais aide aussi à améliorer la qualité des données.

Le profilage d'une seule colonne : ici on se concentrera sur l'extraction d'un maximum d'informations de chaque colonne. Ces informations peuvent être : les types atomiques (NUMBER, STRING, DATE), la cardinalité (nombre de lignes, nombre de valeurs distinctes, ...), le type sémantique (nom, prénom, marque de voiture), ou des informations plus complexes permettant d'identifier la colonne par rapport à d'autres colonnes.

Le profilage de plusieurs colonnes consiste à extraire des métadonnées de plusieurs colonnes simultanément. Ces métadonnées peuvent décrire les liens inter-colonnes avec par exemple, l'extraction de dépendances fonctionnelles ou tout autre type de contraintes.

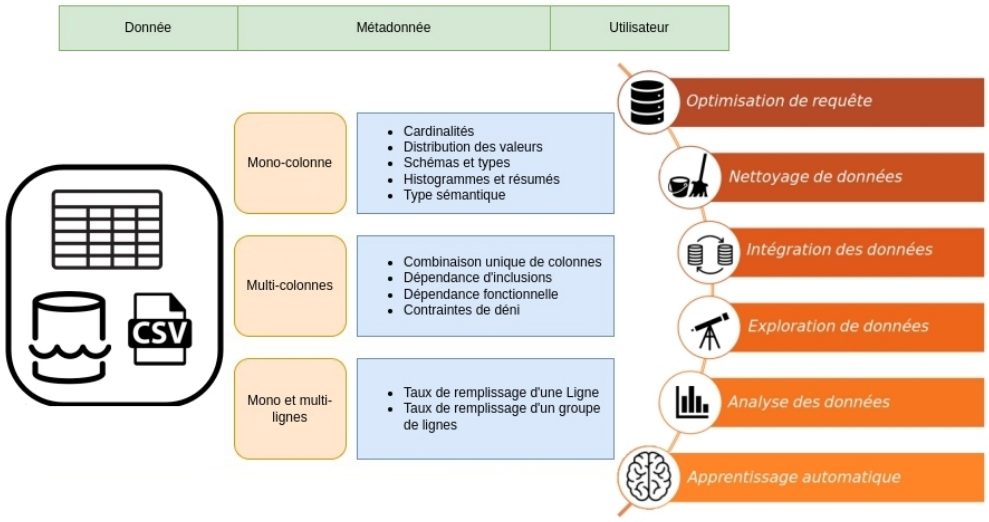


FIGURE 2.1 – Aperçu des métadonnées et de leurs utilisations [Har20].

2.2.1 Profilage des données mono-colonnes

L'étape la plus courante du profilage des données est l'analyse des colonnes une par une, principalement en considérant les colonnes comme in-

dépendantes les unes des autres. Ce type de profilage permet d'extraire des informations très simples comme le nombre de valeurs manquantes (NULL VALUES ou MISSING VALUES) mais peut aussi concerner des informations beaucoup plus complexes comme le type sémantique des colonnes. Le profilage peut être effectué de manière exacte ou de manière approchée en fonction de la volumétrie des données à traiter et du temps disponible pour le profilage.

Catégorie	Description
Cardinalité	Nombre de lignes Nombre ou ratio de valeurs nulles Nombre de valeurs distinctes Nombre de valeurs distinctes divisé par le nombre de lignes
Distributions de valeurs	Histogrammes de fréquences Valeur minimum et maximum dans les colonnes numériques Fréquence du mode divisé par le nombre de lignes Quartile pour les colonnes numériques Distribution du chiffre en première position pour les colonnes numériques
Types de données Schéma Domaines	Type basique (atomique) : Numérique, alphanumérique, date Type spécifique de base de données Statistique sur la longueur des chaines de caractères Taille des chiffres Nombre de décimales Types sémantiques génériques : code, texte, date, quantités Types sémantiques complexes : numéro de carte bancaire, prénom, nom de ville

FIGURE 2.2 – Les profilages des données mono-colonnes

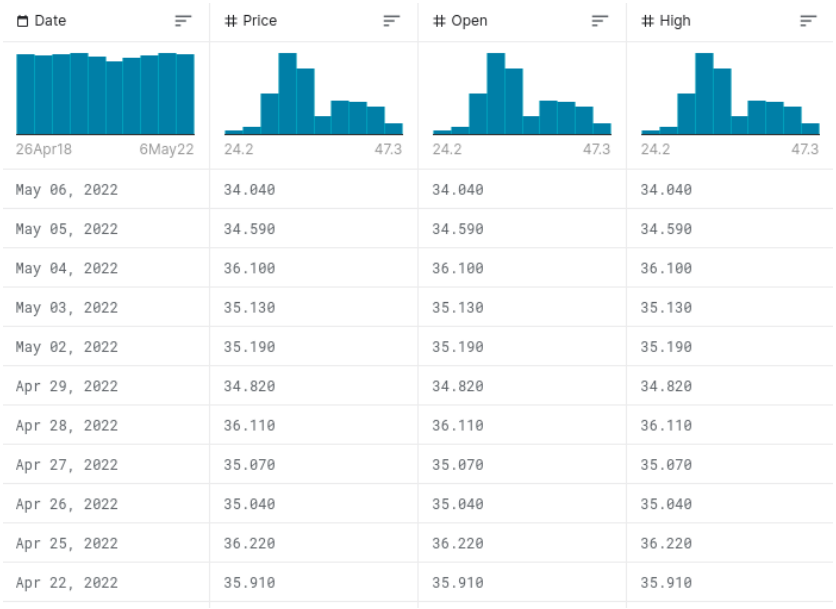
2.2.1.1 Cardinalité

Ce type de métadonnées correspond à des valeurs simples issues de décomptes ou de ratios, qui vont permettre de résumer simplement les données. Le nombre de valeurs distinctes aussi appelé moment de fréquence 0, cette valeur est importante pour l'optimisation des requêtes et si on la divise par le nombre total de valeurs dans la colonne on obtient "l'unicité". Compter le nombre de valeurs manquantes dans chaque colonne permet d'évaluer la complétude du jeu de données. Cependant ce décompte ne permet que de cerner partiellement la complétude des données. En effet, il faut aussi prendre en compte les valeurs par défaut qui peuvent exister dans la colonne, comme par exemple la première valeur d'un formulaire ou un code postal (français) mis à 00000 ou 99999. Ces valeurs ne sont pas toujours aisées à identifier et il faut souvent l'expertise d'un data analyste pour les identifier. Cependant

des algorithmes récents comme FAHES [QECF+18] permettent de détecter certaines valeurs par défaut non triviales.

2.2.1.2 Distribution de valeurs

Une autre tache classique du profilage des données consiste à résumer les données à l'aide d'une représentation plus simple à visualiser. La méthode traditionnelle pour effectuer cette représentation consiste à créer un histogramme avec des bandes de largeurs fixes qui vont représenter des descomptes des valeurs de la colonne. Un exemple de cette méthode est présenté dans la figure 2.3 sur le site de Kaggle [1] (une source de données en accès libre -open access-).



Date	Price	Open	High
May 06, 2022	34.040	34.040	34.040
May 05, 2022	34.590	34.590	34.590
May 04, 2022	36.100	36.100	36.100
May 03, 2022	35.130	35.130	35.130
May 02, 2022	35.190	35.190	35.190
Apr 29, 2022	34.820	34.820	34.820
Apr 28, 2022	36.110	36.110	36.110
Apr 27, 2022	35.070	35.070	35.070
Apr 26, 2022	35.040	35.040	35.040
Apr 25, 2022	36.220	36.220	36.220
Apr 22, 2022	35.910	35.910	35.910

FIGURE 2.3 – Un exemple d'histogramme sur kaggle

La plupart des outils de préparation des données (par exemple trifacta [2]) proposent ce type de représentation. Des méthodes plus avancées utilisent des histogrammes biaisés [CKMS06] afin d'essayer de proposer une représentation plus fidèle des données. Ce type de représentation est particulièrement utile aux analystes de données, elle leur permet de visualiser rapidement la répartition des données afin d'identifier le type de distribution qu'elles suivent ou la présence de données aberrantes.
Le calcul des extremums des valeur numériques permet l'identification de données aberrantes qui se trouveraient en dehors de la plage de valeurs attendues.

On peut définir un indicateur clé appelé constance de la colonne qui va correspondre à la fréquence du mode (valeur la plus fréquente dans la co-

1. www.kaggle.com/datasets/elgunisgandarli/government-bonds-etfs-and-funds
2. www.trifacta.com/fr

lonne) divisée par le nombre total de valeurs dans la colonne.

Il est intéressant pour les colonnes numériques de vérifier si les données respectent la loi de Benford qui stipule que la probabilité de présence dans une colonne d'un digit d en première position d'une valeur numérique est : $P(d) = log_{10}(1 + d^{-1})$ [Ben38]. On va ainsi s'attendre à une très grande présence de 1 et une faible présence de 9.

Il est aussi possible d'extraire des indicateurs plus complexes comme l'entropie de la distribution des valeurs [KN03], le décompte des cases contenants des caractères spéciaux, la moyenne du comptage du nombre de chaque caractère dans chaque case, les possibilités sont infinies. Un grand nombre d'indicateurs de ce type seront utilisés dans le chapitre 3 afin de construire un vecteur d'indicateurs qui représentent une colonne.

Enfin, on peut calculer des indicateurs non pas directement sur les données mais sur une version transformée des données, par exemple une analyse de la fréquence des codes soundex [Knu73]. Soundex est une méthode permettant d'encoder une chaîne de caractères selon sa phonétique souvent utilisée pour comparer deux chaînes de caractères afin de saisir des informations différentes des méthodes comparant les caractères [Nav01].

2.2.1.3 Types de données, schémas, domaines

Dans cette section nous allons discuter de la recherche du type atomique ou basique des colonnes, des types complexes comme les types sémantiques mais aussi de la recherche de schémas récurrents dans les données.

Les types atomiques ou basiques sont par exemple : numérique, alphabétique, alphanumérique, date. Les trois premiers sont facilement identifiables en observant la présence ou l'absence de caractère numérique ou non numérique. Les dates sont plus difficiles à détecter mais peuvent l'être en vérifiant certaines plages de valeurs numériques et certains motifs réguliers se répétant comme XX/XX/XX. De plus il existe un grand nombre de possibilités de formats de date mais ces format sont tous détectables à l'aide d'expressions régulières.

Ces types atomiques peuvent être raffinés avec des types plus précis comme entier ou booléen. Ces types peuvent être vérifiés en observant la présence ou non de décimales, ou en comptant la présence ou l'absence de certains caractères numériques.

Une autre activité de profilage consiste à détecter des motifs récurrents dans les données. Cette activité est simple quand les motifs sont réguliers et

les formats connus, on va ainsi pouvoir chercher des numéros de téléphone, des adresses mails ou tout type d'identifiant ayant une structure bien définie. Le problème devient beaucoup plus complexe quand il s'agit de découvrir de manière automatique des motifs dans les données. En effet, le motif ne doit pas être trop généraliste car sinon il n'apporte aucune information et au contraire s'il est trop restrictif il manque de l'information. Ainsi au delà de plusieurs centaines de lignes il n'est pas possible de découvrir automatiquement des expressions régulières en un temps raisonnable [IdTFM18].

Une information complexe à obtenir est le type sémantique (ou domaine sémantique des données). C'est une information utile pour de nombreux autres domaines comme le nettoyage et la transformation de données [KPHH11], l'enrichissement de caractéristiques [GKH+19] ou encore la recherche dans des données structurées [GK20].

Traditionnellement la recherche du type sémantique se fait à l'aide d'une combinaison d'expressions régulières et de table de correspondances. Cette méthode est efficace sur les types sémantiques pouvant être décrits par un motif très régulier (comme des adresses mail) ou ayant un faible nombre fini d'éléments (comme le nom des continents). Cependant ces méthodes ne sont applicables qu'à un nombre limité de types sémantiques et peuvent avoir de mauvais résultats si les données contiennent des erreurs. D'autres méthodes se basent sur des fonctions de similarités floues pour être plus résilientes aux erreurs dans les données [DJL+13]. Ces méthodes présentent aussi des désavantages comme la confusion entre types sémantiques ayant des valeurs communes ou proches.

Pour pallier ces inconvénients, de nouvelles approches utilisant l'apprentissage automatique sont apparues ces dernières années [SLL+22, ZHS+20, TCH21]. L'une des méthodes les plus connues est Sherlock [HHB+19], qui traite le problème de la détection du type sémantique des données comme un problème de classification multi-classes. De chaque colonne, Sherlock extrait 1588 caractéristiques réparties en 4 grands types.
Les caractéristiques de type "Statistiques globales" qui décrivent comment les valeurs sont distribuées dans la colonne. Les caractéristiques de type "distribution de caractères" qui décrivent comment les caractères sont distribués dans les colonnes, et enfin, les caractéristiques des types "plongement de mots" [PSM14] et "paragraphes vectorisés" [LM14] qui utilisent deux techniques de plongement de mots pour décrire les colonnes de deux manières différentes. Un réseau neuronal profond est ensuite entraîné à l'aide d'un grand volume de données afin d'apprendre à distinguer les classes. Cette méthode est plus résistante aux erreurs dans les données (mauvais caractères, éléments n'appartenant pas au domaine) et donne de meilleurs résultats que les méthodes classiques. La méthode de Sherlock a donné lieu à des raffi-

nements, notamment [ZHS+20]. SATO, considère le problème comme un problème de prédiction multi-colonnes. C'est-à-dire qu'il prédit les types sémantiques de toutes les colonnes d'une table en une seule fois. Ceci est fait dans le but d'améliorer les prédictions en apprenant les relations qui peuvent exister entre les colonnes (par exemple : une colonne de prénoms est souvent accompagnée d'une colonne de noms). Nous avons fait le choix de nous intéresser particulièrement à ce type de méthode, car les données que nous envisageons de traiter sont généralement de mauvaise qualité et contiennent beaucoup de bruits.

2.2.1.4 Approximations

L'accumulation de grandes quantités de données rend le profilage et le calcul de métadonnées à priori simple, long et coûteux. Il devient alors nécessaire d'adopter des stratégies de calcul approximatif afin de réduire les coûts en calculs.
La méthode la plus simple pour effectuer des calculs approximatifs consiste à échantillonner uniformément les données : cela va permettre d'obtenir une distribution approximative des données (ne convenant pas à l'extraction précise de toutes les métadonnées) qui va permettre de construire des histogrammes, d'estimer le nombre de valeurs distinctes (de manière peu précise), de prédire le type basique des données, de repérer des motifs récurrents dans les données [HNSS95].
Une autre possibilité est d'essayer de résumer de manière très simple les données à l'aide de compteurs qui vont s'incrémenter quand une valeur connue est rencontrée et se décrémenter quand une valeur non encore vue est rencontrée [MG82]. Enfin, il existe un grand volume de littérature [Har20] sur des méthodes dans lesquelles les données sont préalablement hachées afin de pouvoir ensuite calculer de manière rapide des métadonnées telles que le nombre de valeurs distinctes.

2.2.2 Profilage des données multi-colonnes

Le profilage des données multi-colonnes porte surtout sur la recherche de dépendances et autres contraintes entre les colonnes. Les dépendances et contraintes peuvent être au choix exactes ou approximatives. Le but est d'essayer d'identifier comment les données sont structurées en identifiant des relations entre les colonnes.

2.2.2.1 Les différents types de contraintes

Nous allons commencer par décrire certains types de contraintes d'intégrité. Tout d'abord les contraintes de déni : elles correspondent à l'un des types de contraintes qui permet le plus d'expressivité.

Les contraintes de déni (en anglais Denial Constraints DC) φ sont définies de la manière suivante :
Si l'on note R le schéma relationnel dans une base de données. I une instance d'un schéma relationnel représentée par une table T ou un fichier CSV. A un attribut de l'instance (une colonne de la table), *dom(A)* l'ensemble des valeurs plausibles pour A. t un tuple ou un enregistrement de T.

$\forall t_\alpha, t_\beta, t_\gamma, ... \in I, \neg(P_1 \wedge ... \wedge P_m)$ où chaque prédicat P_i est de la forme $v_1 \theta v_2$ ou $v_1 \theta c$ avec $v_1, v_2 \in t_x.A$, $x \in \{\alpha, \beta, \gamma, ...\}$, $A \in R$, c une constante dans *dom(A)* et $\theta \in \{=, <, >, \leq, \geq, \neq\}$.

La table 2.1 nous servira d'exemple pour illustrer le fonctionnement des contraintes de déni. Les données sont celles d'un service de livraison. PR représente la présence ou non lors de la livraison, NCL le numéro du client, NCO le numéro de commande, NDEP le numéro du département, DEP le nom du département, VI la ville, TVA le pourcentage de tva sur les articles, HT le prix hors taxes, TTC le prix toutes taxes comprises.

TABLE 2.1 – Exemple de données client

ID	PR	NCL	NCO	NDEP	DEP	VI	TVA	HT	TTC
t1	O	C12321	1372	75	paris	paris	5	123	129,15
t2	O	C37487	123	75	paris	paris	12	567	635,04
t3	N	C34532	2134	60	oise	creil	10	123	135,3
t4	N	C45454	28	75	paris	paris	5	12	12,6
t5	O	C47597	32	75	paris	paris	12	89	99,68
t6	O	C87970	54	60	oise	beauvais	12	45	50,4
t7	N	C37487	233	60	oise	creil	5	345	362,25
t8	N	C65486	1372	80	somme	abbeville	5	89	93,45
t9	N	C34234	34	80	somme	amiens	5	89	93,45

Nous pouvons à partir de ces données définir plusieurs contraintes de déni :

$$d_{c_0} : \forall t_\alpha \in I, \neg(t_\alpha.PR \neq O \wedge t_\alpha.PR \neq N) \tag{2.1}$$

Cette contrainte 2.1 stipule que PR ne peut prendre que les valeurs O ou N. Une personne est soit présente soit absente de son domicile.

$$d_{c_1} : \forall t_\alpha, t_\beta \in I, \neg(t_\alpha.NCL = t_\beta.NCL \wedge t_\alpha.NCO = t_\beta.NCO) \tag{2.2}$$

La contrainte 2.2 exprime le fait qu'un couple numéro de commande et

numéro de client est unique. Un client peut avoir plusieurs numéros de commande, et un numéro de commande peut être utilisé pour plusieurs clients différents. Mais en aucun cas un même client ne peut avoir plusieurs fois le même numéro de commande.

$$d_{c_2} : \forall t_\alpha, t_\beta \in I, \neg(t_\alpha.NDEP = t_\beta.DEP \wedge t_\alpha.DEP \neq t_\beta.DEP) \quad (2.3)$$

La contrainte 2.3 exprime le fait qu'un numéro de département n'est attaché qu'à un seul département et qu'un département n'a qu'un seul numéro de département.

$$d_{c_3} : \forall t_\alpha \in I, \neg(t_\alpha.VI = "paris" \wedge t_\alpha.NDEP \neq "75") \quad (2.4)$$

La contrainte 2.4 exprime le fait que la seule ville du département 75 est Paris.

$$d_{c_4} : \forall t_\alpha, t_\beta \in I, \neg(t_\alpha.TVA = t_\beta.TVA \wedge t_\alpha.HT < t_\beta.HT \wedge t_\alpha.TTC < t_\beta.TCC) \quad (2.5)$$

La contrainte 2.5 nous informe que si deux articles ont la même TVA, celui qui a le prix hors taxes le plus faible, aura le prix TTC le plus faible.

$$d_{c_5} : \forall t_\alpha \in I, \neg(t_\alpha.HT < t_\alpha.TTC) \quad (2.6)$$

Enfin, la contrainte 2.6 indique que le prix HT est toujours inférieur au prix TTC. Ces exemples sont intéressants car ils permettent d'illustrer le nombre incroyable de possibilités qu'offre le cadre des contraintes de déni.

En notant *attr(R)* l'ensemble des attributs de R, une dépendance fonctionnelle (Functional Dependency DF) φ est définie par X→Y où X $\subseteq$ *attr(R)* et $Y \subseteq$ *attr(R)*. Une instance I de R satisfait une DF φ notée $I \models \varphi$ si pour chaque duo de tuples t_α, t_β dans I tels que $t_\alpha[X] = t_\alpha$[X] on a $t_\alpha[Y] = t_\alpha$[Y]. On s'intéressera uniquement au cas où la dépendance fonctionnelle est dite minimale. Une dépendance fonctionnelle X→Y est dite minimale (sur I) si aucun sous-ensemble de X ne forme de dépendance fonctionnelle avec Y. De plus, on s'intéressera uniquement au cas où l'ensemble Y est de taille 1 (sinon la dépendance est dite multivaluée).

Par exemple dans le tableau 2.2 nous pouvons identifier les dépendances fonctionnelles suivantes : Prénom→Genre (sur cette instance) et {Jour N, Mois}→Signe. Dans la pratique ce type de dépendance est le plus recherché.

Enfin on peut remarquer qu'une dépendance fonctionnelle est un cas particulier de DC, par exemple la DC 2.3 est une dépendance fonctionnelle. De

TABLE 2.2 – Exemple de données Signe astrologique

Prénom	Genre	Jour N	Mois	Signe
Marc	M	12	Juin	Gémeaux
Prune	F	14	Juillet	Cancer
Paul	M	28	Août	Vierge
Cerise	F	12	Juin	Gémeaux
Marc	M	25	Mars	Bélier
Prune	F	25	Mars	Bélier

plus les exemples que nous avons donnés se concentrent sur la détection de contraintes exactes, il est tout à fait possible de s'intéresser à des versions approximatives où la contrainte n'est respectée que sur un sous-ensemble des données. Ce qui, dans le cadre des dépendances fonctionnelles, correspond à la notion de dépendance fonctionnelle partielle [AGN15] qui se note $X \vdash_{\psi \leq \epsilon} Y$ et où la dépendance n'est vérifiée que si le taux d'erreur lors de la vérification ϵ est inférieur à un seuil ψ [CDP16]. Ce type de dépendance peut être encore approfondi avec les dépendances fonctionnelles conditionnelles [FGJK08], dans lesquelles on précise les conditions pour lesquelles la dépendance doit être vérifiée ou non.

Il est possible d'élargir le choix de fonction pour θ (pour les DC) afin de permettre encore plus d'expressivité. Cela mène au niveau des dépendances fonctionnelles au concept de "matching dependency" ou l'égalité est remplacée par une fonction de similarité [Fan08].

La recherche de dépendances est souvent visualisée sous la forme d'une treillis comme dans la figure 2.4 qui représente les dépendance fonctionnelles possibles pour un attribut F.

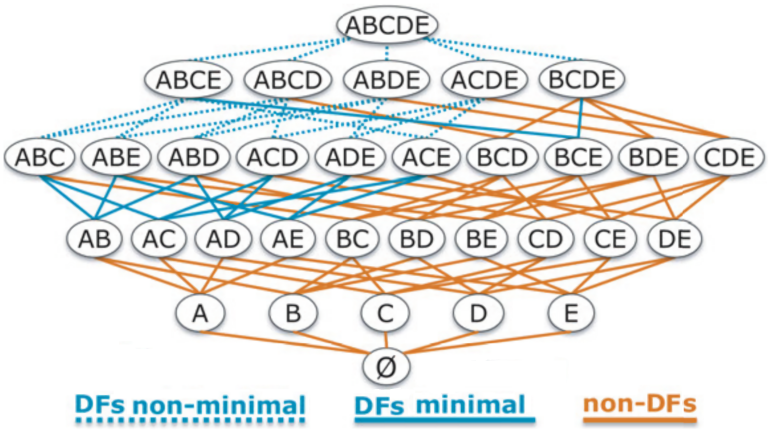


FIGURE 2.4 – Treillis de recherche de DF [AGNP18]

Il existe un très grand nombre d'algorithmes de détection de dépendances

fonctionnelles explorant de manière différente l'espace de recherche [HKPT99, WGR01, PEM+15]. La plupart des algorithmes récents s'appuient sur une structure de données nommée liste d'indice de position notée PLI. Cette structure permet de transformer le problème de vérification des DF en un problème d'intersection d'ensemble. De plus les algorithmes les plus récents utilisent aussi des méthodes d'échantillonage pour réduire le temps de calcul [PN16]. Cependant si l'on note n le nombre de lignes et m le nombre d'attributs la complexité $\mathcal{O}(n^2 * 2^m * (\frac{m}{2})^2)$ ce qui rend la recherche très difficile en un temps raisonnable au delà d'une vingtaine d'attributs. De même pour la recherche des DC, si l'on considère les 6 opérateurs : $\{=, <, >, \leq, \geq, \neq\}$ la taille de l'espace de recherche est de l'ordre de $2^{(6*2m*(2m-1))} \approx 2^{m^2}$ [CIP13, BKN17] ce qui rend la recherche des DC extrêmement difficile.

2.2.3 Profilage des données Mono et Multi-lignes

Le profilage des lignes est un thème peu abordé dans la littérature, il apporte pourtant un éclairage différent sur les données. Ce type de profilage consiste à calculer des statistiques sur chacune des lignes ou sur un groupe de lignes au lieu de les calculer sur les colonnes. On va par exemple calculer le taux de remplissage d'une ligne en divisant le nombre de valeurs manquantes dans une ligne par le nombre de colonnes. Ce taux peut permettre d'éliminer des lignes très peu remplies des autres traitements en les jugeant aberrantes. En effet, en comparant le taux de remplissage d'une ligne par rapport à celui des autres on peut identifier les lignes aberrantes. Un exemple est présenté dans le tableau 2.3. La ligne 6 ayant un faible taux de remplissage, il est possible de la considérer comme aberrante.

TABLE 2.3 – Exemple de profilage de ligne

ligne	genre	prénom	âge	taille	ville	taux de remplissage
1	M	Paul	63	NULL	Paris	0.8
2	F	Jeannne	45	165	Vincennes	1
3	M	NULL	23	194	Paris	0.8
4	F	Astrid	33	155	Paris	1
5	M	marc	67	167	NULL	0.8
6	NULL	Karl	NULL	NULL	NULL	**0.2**

La plupart des statistiques calculables sur une colonne peuvent être adaptées aux lignes, nous pouvons par exemple dénombrer le nombre de valeurs numériques ou alphanumériques dans une ligne. Pour les statistiques multi-lignes le principe est de découper les données tabulaires en groupe de lignes puis moyenner les résultats individuels des lignes pour créer les résultats finaux du groupe de lignes.

2.2.4 Exemples d'outils de profilage

Le Schéma suivant présente les principaux usages du profilage des données.

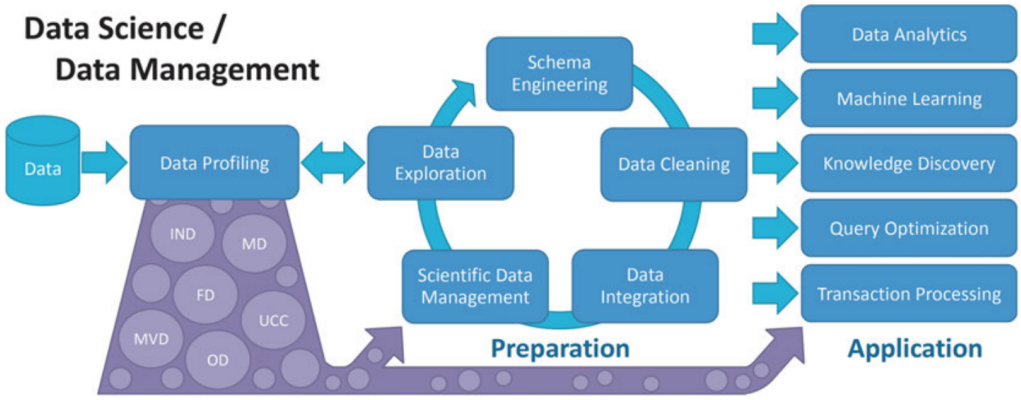


FIGURE 2.5 – Exemple d'usage du profilage des données [AGNP18]

Ainsi les principaux domaines utilisant le profilages des données sont les suivant.

L'exploration des données qui consiste à étudier la structure et la sémantique des données avant d'en faire usage. Cela permet à une personne de mieux comprendre les données qu'elle doit traiter, c'est une étape importante de la préparation des données. Ceci s'avère particulièrement compliqué quand les données sont de grande dimension. Dans ce cas, l'usage du profilage permet d'appréhender les données à partir d'un résumé de celles-ci. Cela permet aussi de cibler les données clés à représenter avant de les visualiser pour approfondir la compréhension des données.

L'intégration des données qui vise à fusionner plusieurs schémas de données entre eux et produire une nouveau schéma contenant les informations les plus importantes des anciens. L'intégration des données peut aussi permettre de regrouper des données provenant de plusieurs sources. Les métadonnées sont alors utiles pour comprendre ce qui peut être ou non fusionné ensemble afin d'obtenir un résultat correct.

Le nettoyage des données qui est une partie appliquée de l'étude de la qualité des données. Ici, les métadonnées sont utilisées pour définir des règles. Les données ne respectant pas ces règles sont ainsi corrigées. Les métadonnées les plus intéressantes seront issues de l'extraction de dépendances ou contraintes approximatives. Car si elles sont exactes, cela implique qu'il n'y a rien à corriger dans les données. Des algorithmes comme Holoclean [RCIR17] proposent des corrections automatiques des données ne respectant pas les contraintes fournies.

Nous allons maintenant abordés quelques outils de profilage des données.

Tout d'abord TRIFACTA [3] :, c'est un outil spécialisé dans la préparation des données. Il rend le profilage très visuel et offre un accès rapide à la plupart des indicateurs de profilage mono-colonne et offre la possibilité de créer des indicateurs personnalisés.

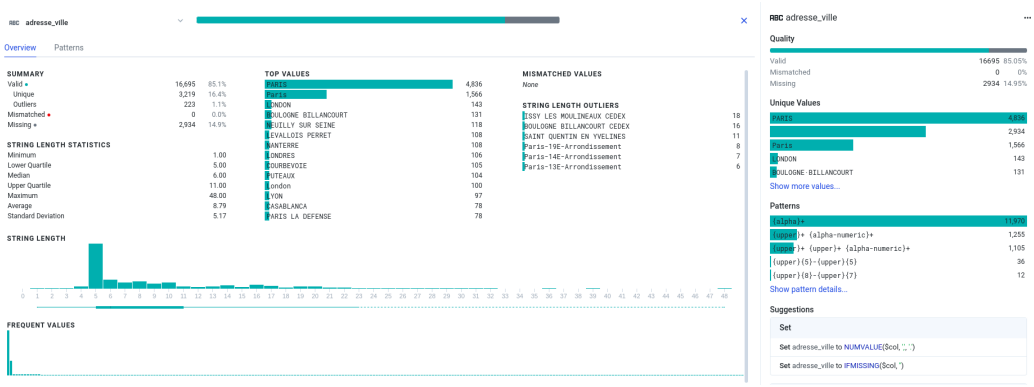


FIGURE 2.6 – Exemple du profilage de données avec TRIFACTA

Column: metadata.Column52

Statistiques simples

Libellé	Nombre	%
Compte de lignes	538	100.00%
Compte null	0	0.00%

Indicateur de fréquence des modèles

Valeur	Nombre	%
99/9/99	460	85.66%
9/99/99	49	9.12%
99/99/99	17	3.17%
9/9/99	6	1.12%
Empty field	2	0.37%
99/99/9999 99:99:99	2	0.37%
aaaaa_aa9	1	0.19%

FIGURE 2.7 – Exemple de résultat proposé par Talend Data Quality

Ensuite, Talend data quality qui est un outil open source utilisable de manière locale pour le profilage des données en vue d'établir des rapports

3. https ://www.trifacta.com/fr/

de qualité. L'outil propose de détecter les schémas les plus représentés dans les colonnes ou de vérifier si certaines colonnes respectent un format précis pré-établi ou définissable par l'utilisateur. De plus, il propose la détection automatique de dépendance fonctionnelle. Cependant, comme Trifacta, cet outil demande beaucoup d'interventions manuelles, ce qui est le plus gros désavantage des outils industriels.

2.3 L'apprentissage artificiel (Machine Learning)

2.3.1 L'apprentissage automatique

L'apprentissage automatique ou machine learning (ML) est un sous-domaine de l'intelligence artificielle qui est elle-même un sous-domaine de l'informatique. L'apprentissage automatique vise à construire des algorithmes qui exploitent des données accumulées qu'elles soient naturelles ou artificielles afin de résoudre certaines problématiques. Les principaux courants du domaine sont : l'apprentissage supervisé, l'apprentissage non supervisé et l'apprentissage par renforcement [Bur19].

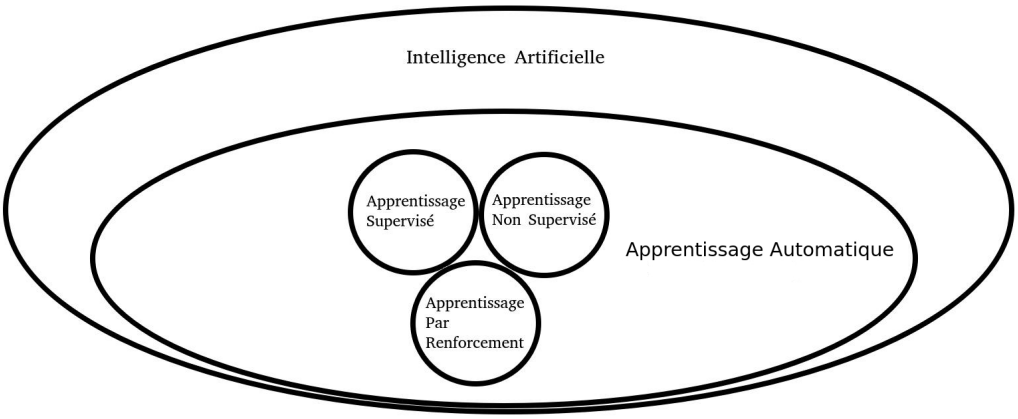


FIGURE 2.8 – Schématisation du domaine de l'intelligence artificielle

Dans l'ensemble de cet ouvrage seuls l'apprentissage supervisé et non-supervisé seront utilisés.

2.3.1.1 L'apprentissage non-supervisé

L'apprentissage non-supervisé regroupe un ensemble de méthodes prenant en entrée des données issues d'une source DS (ces données sont généralement représentées sous forme de vecteurs dans un jeu de données) et les transforme en un autre vecteur ou une valeur servant pour résoudre un problème. L'apprentissage non-supervisé est généralement utilisé pour réduire la dimension des vecteurs issus de la source DS. Ceci permet de rendre plus facilement

visualisable les données de grande dimension [vdMH08] ou de réduire l'effet de la malédiction des grandes dimensions [BHK20]. Mais ces algorithmes peuvent aussi servir à regrouper les données similaires en groupes [JH10] ou détecter des données aberrantes [Agg16].

2.3.1.2 L'apprentissage par renforcement

L'apprentissage par renforcement est un type particulier d'apprentissage. Le but est d'apprendre à effectuer un ensemble de décisions de manière séquentielle. La fonction que l'on cherche à apprendre s'appelle une politique. L'agent passe par un certain nombre d'états.
Dans chacun de ces états, il doit effectuer une action. Chaque action implique une forme de récompense et peut mener à un changement d'état. Le but est donc d'apprendre la suite d'actions qui va maximiser les récompenses reçues par l'agent. Ce type d'apprentissage est utilisé en robotique et dans les jeux. Des applications existent aussi dans le monde de la correction des données comme par exemple le projet learn2clean [BE19]. Dans ce projet l'algorithme apprend à déterminer selon une métrique de performance choisie, pour un algorithme d'apprentissage automatique choisi, la meilleure suite d'actions de préparation et de nettoyage des données à effectuer.

2.3.1.3 L'apprentissage Supervisé

L'apprentissage supervisé peut être décrit de la façon suivante (fig 2.10) :

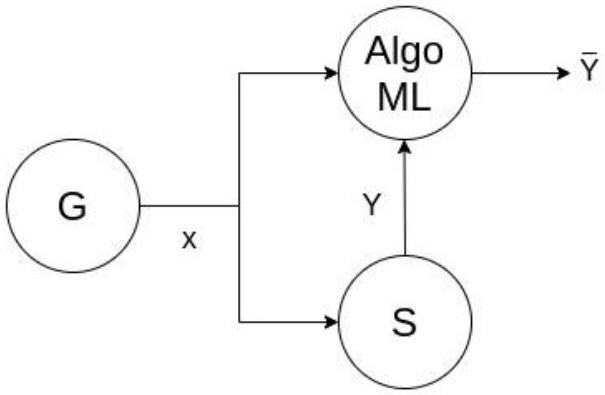


FIGURE 2.9 – Représentation de l'apprentissage supervisé

Un générateur G qui produit des données (N valeurs) sous forme de vecteurs de caractéristiques x_i. Ce générateur peut être aussi bien physique qu'artificiel [VVV98].

Un opérateur inconnu nommé superviseur S vient associer à chaque x_i un label y_i. Ce label peut être issu d'un ensemble fini de classes, mais il peut aussi être une valeur réelle, un graphe, une matrice ou encore un vecteur. L'ensemble des couples $\{(x_i, y_i)\}_i^N$ forme un jeu de données.

L'algorithme de ML doit pour chaque x_i réussir à approximer y_i et ainsi produire un résultat $\overline{y_i}$, le but est alors d'imiter au mieux la réponse du superviseur.

Quand y_i prend des valeurs discrètes et finies on parle d'un problème de classification et l'algorithme de ML est communément appelé un classifieur. Par exemple, si l'on a un jeu de données contenant des images de chiens et de chats et que l'on cherche à prédire quelle image est celle d'un chien ou celle d'un chat. Quand y_i prend des valeurs continues on parle généralement de régression. Par exemple en connaissant des informations sur la surface et la localisation d'une maison prédire une estimation de son prix.
Dans le monde de la qualité des données on peut utiliser des algorithmes supervisés pour apprendre à distinguer entre des données normales et des données aberrantes.

Dans l'ensemble de ce document on ne s'intéressera qu'au problème de classification. Les principaux classifieurs que nous avons utilisés sont représentés dans la figure 2.10.

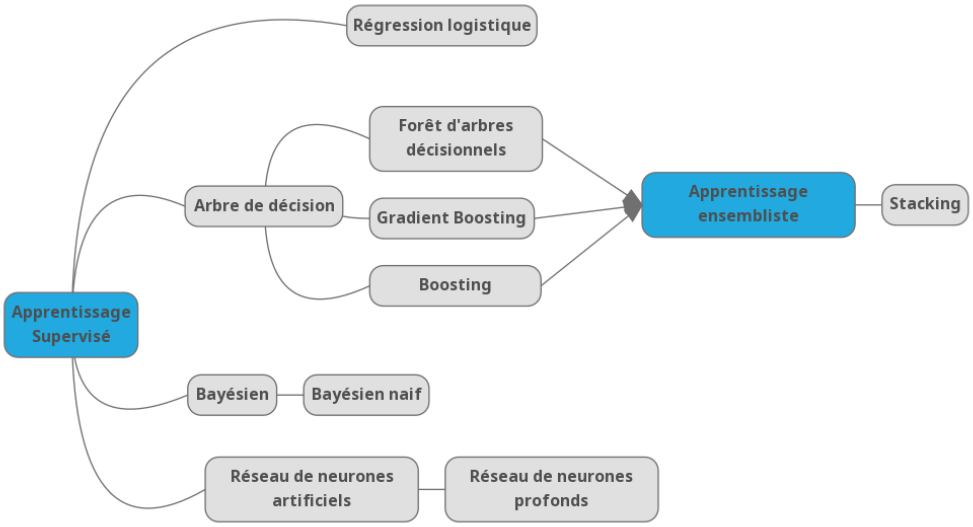


FIGURE 2.10 – Les types de modèles que nous avons utilisés

Afin de pouvoir comparer les performances de plusieurs algorithmes d'apprentissage il faut pouvoir les comparer. Pour ceci on va définir des métriques qui vont nous permettre d'évaluer les performances d'un classifieur sur un jeu de données. Les définitions que nous allons donner sont dans le cas binaire mais peuvent s'étendre aux problèmes multi-classes [GBV20]. Dans le cas binaire, une méthode pour représenter les résultats est la matrice de confusion, celle-ci permet de visualiser quels types d'erreurs commet le classifieur.

La première mesure est la précision :

$$Précision = \frac{VP}{VP + FP} \tag{2.7}$$

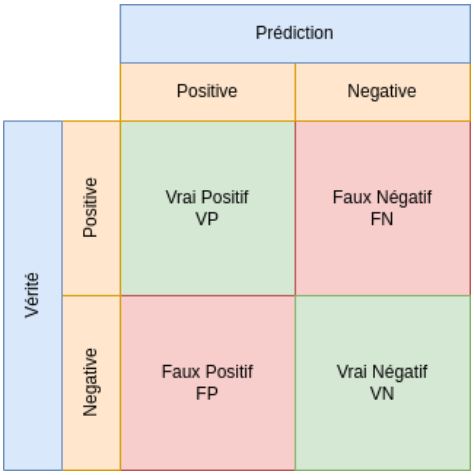


FIGURE 2.11 – Matrice de confusion binaire

La précision se définit par le nombre d'individus correctement prédits (VP pour vrai positif) divisé par le nombre total d'éléments prédits comme étant positifs (FP pour faux positif). Cette mesure va nous permettre d'évaluer la confiance que l'on peut avoir dans les prédictions positives d'un classifieur. C'est une mesure importante quand le coût d'un faux positif est très élevé.

$$Rappel = \frac{VP}{VP + FN} \tag{2.8}$$

Le rappel se définit par le nombre d'individus correctement prédits divisé par le nombre d'éléments à prédire comme positifs. Cette mesure est particulièrement intéressante à utiliser quand les faux négatifs ont d'importantes conséquences.

La mesure suivante, le f1-score combine les deux précédentes :

$$f1 - score = 2 * \frac{Précision + Rappel}{Précision * Rappel} \tag{2.9}$$

Le f1-score prenant des valeurs entre 0 et 1 est une mesure qui est utilisée quand l'on considère que le rappel et la précision ont la même importance et que l'on cherche à trouver le meilleur compromis entre précision et rappel.

Enfin, la mesure la plus couramment utilisée est le taux de bonne reconnaissance ; il se définit par :

$$Taux\ de\ bonne\ reconnaissance = \frac{VP + VN}{VP + VN + FP + FN} \tag{2.10}$$

Cette formule correspond à la division du nombre d'individus correctement prédits par le total d'individus. Cette mesure permet de rapidement identifier si le classifieur prédit correctement. Cependant, il est à noter qu'elle

n'est vraiment intéressante à utiliser que lorsque les classes sont équilibrées (un nombre similaire d'individus dans chaque classe). En effet, quand un déséquilibre est présent, le taux de bonne reconnaissance favorise la classe la plus présente. Dans ce genre de scénario il vaut mieux utiliser le f1-score.

Classiquement pour évaluer un modèle sur un jeu de données on va découper le jeu de donnée en trois sous-ensembles, l'un d'apprentissage, l'un de test et le dernier pour la validation. Cependant, quand on désire avoir des résultats plus robustes, on va appliquer une technique s'appellent la validation croisée. Le jeu de données va être découpé en k-blocs ou k-plis, k - 1 plis sont utilisés pour l'apprentissage et un pli pour le test. On recommence cette opération k fois afin d'obtenir k mesures. On peut ainsi exploiter comme résultat final la moyenne ainsi que l'écart type des k mesures. Le fonctionnement de la méthode est décrit dans la figure 2.12.

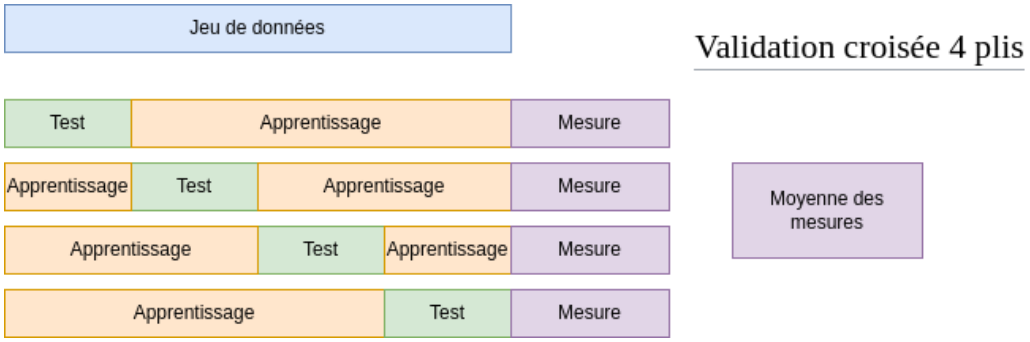


FIGURE 2.12 – Validation croisée 4 plis

2.3.2 Les modèles supervisés que nous avons utilisés

Cette section va présenter les modèles que nous avons utilisés dans ce document et qui sont représentés dans la figure 2.10.

2.3.2.1 Classifieur bayésien naïf

Un classifieur bayésien, est un classifieur probabiliste se basant sur le théorème de bayes pour faire ses prédictions. On va chercher à estimer la probabilité qu'un individu appartienne à une classe Q sachant ses attributs $X = (x_1, x_2, ...x_n)$. Le théorème de base nous donne :

$$P(Q_k|X) = \frac{P(X|Q_k)P(Q_k)}{P(X)} \; pour \; k \; = \; 1, 2, .., K \tag{2.11}$$

Ainsi le problème de classification s'écrit :

$$\underset{Q_k}{\text{Argmax}} \frac{P(Q_k)P(X|Q_k)}{P(X)} \tag{2.12}$$

$P(Q_k)$ se nomme la probabilité à priori de la classe, $P(X|Q_k)$ la vraisemblance et $P(Q_k|X)$ la probabilité à posteriori. $P(Q_k)$ appelée évidence est une constante.

Dans le cas naïf on va supposer que l'existence d'une caractéristique pour une classe, est indépendante des autres caractéristiques. Ce qui va se traduire par :

$$P(x_i|x_j, Q) = P(x_i|Q), \forall i \neq j \tag{2.13}$$

Ce qui donne dans la formule 2.11 :

$$On\ a\ :\ P(Q_k|X) = \frac{P(Q_k)P(x_1|Q_k)P(x_2|Q_k)..., P(x_n|Q_k)}{P(x_1, x_2, ..., x_n)} \tag{2.14}$$

$P(x_1, x_2, ..., x_n)$ étant constant on cherche à déterminer :

$$\underset{Q_k}{\text{Argmax}}\, P(Q_k) \prod_{i=1}^{n} P(x_i|Q_k) \tag{2.15}$$

Il existe plusieurs types de classifieurs bayesiens naïfs. On utilisera dans ce document celui de type gaussien qui prend pour hypothèse que les caractéristiques suivent une loi normale. Ce classifieur est utilisé tout au long du chapitre 4.

2.3.2.2 Arbres décisionnels

Un arbre de décisions est un graphe acyclique qui va permettre de prendre des décisions. Nous allons ici décrire le cas d'un arbre de décisions binaire pour la classification en utilisant les données du tableau 2.4 le but étant de prédire si le pied est cassé ou non à l'aide des autres colonnes.

TABLE 2.4 – Exemple de jeu de données pour les arbres de décisions

ligne	Douleur au pied	Peut Bouger le pied	Pied Gonflé	Pied cassé
1	Non	Oui	Non	Oui
2	Oui	Non	Oui	Oui
3	Oui	Oui	Non	Non
4	Oui	Non	Oui	Oui
5	Oui	Non	Oui	Oui
6	Non	Oui	Oui	Non
7	Non	Oui	Non	Non
8	Oui	Non	Oui	Oui
9	Oui	Oui	Non	Non
10	Oui	Oui	Oui	Oui

Pour construire l'arbre, nous devons commencer par la racine de l'arbre qui contiendra toutes nos données. Puis les données seront peu à peu séparées à l'aide des caractéristiques. Chaque séparation donne naissance soit à un nouveau noeud séparateur ou à un noeud terminal (appelé feuille).

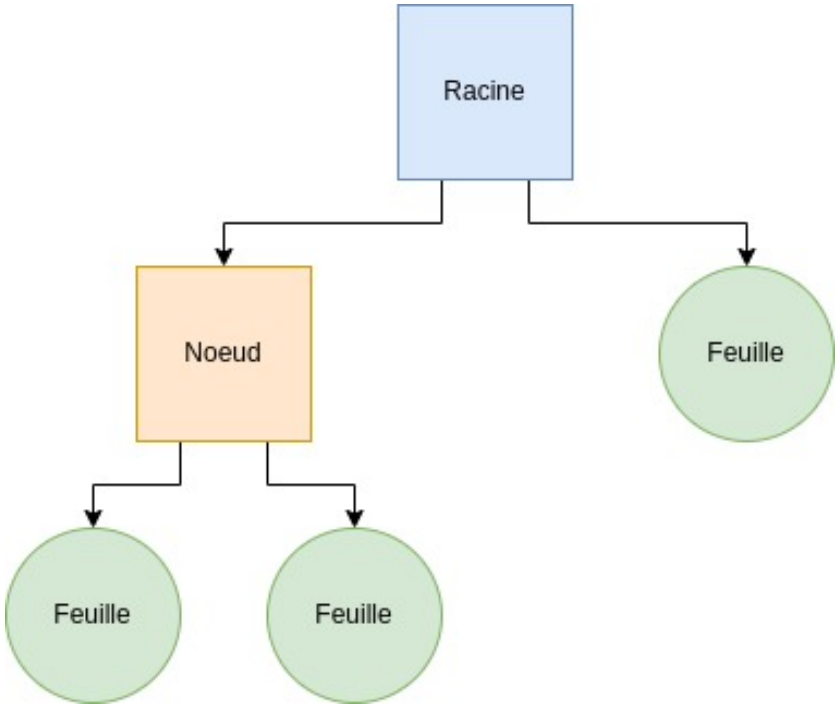


FIGURE 2.13 – Arbre de décisions

Ensuite nous allons choisir parmi nos 3 caractéristiques celle qui séparera le mieux nos données en deux sous-groupes. Le critère que nous allons utiliser pour choisir le meilleur candidat pour la séparation est l'indice de diversité de Gini.

$$Gini = \sum_{i=1}^{Q} P(i) * (1 - P(i)) = 1 - \sum_{i=1}^{Q} P(i)^2 \tag{2.16}$$

Pour déterminer quelle est la meilleure caractéristique à choisir, on va calculer son indice de diversité de Gini, celui-ci correspond à la moyenne pondérée des indices des deux choix possibles. On va effectuer ce calcul pour chacune des caractéristiques possibles puis conserver la caractéristique qui aura l'indice le plus bas. Les calculs suivants détaillent la méthode pour la racine :

$G_{douleurpied}(Oui) = 1 - (\frac{5}{7})^2 - (\frac{2}{7})^2 = 0.4081$

$G_{douleurpied}(Non) = 1 - (\frac{1}{3})^2 - (\frac{2}{3})^2 = 0.4444$

$G_{douleurpied} = (\frac{7}{7+3}) * 0.4081 + (\frac{3}{7+3}) * 0.4444 = 0.4189$

De même on a :

$G_{Peutbouger} = 0.26$
$G_{Piedgonflé} = 0.3166$

On va donc choisir la valeur la plus faible, ici cela correspond à la caractéristique "peut bouger". Nous séparons ainsi notre population initiale en deux. Puis nous recommençons le processus de sélection d'une variable séparatrice à l'aide du calcul de l'indice de diversité de Gini. La figure 2.14 détaille l'arbre totalement construit.

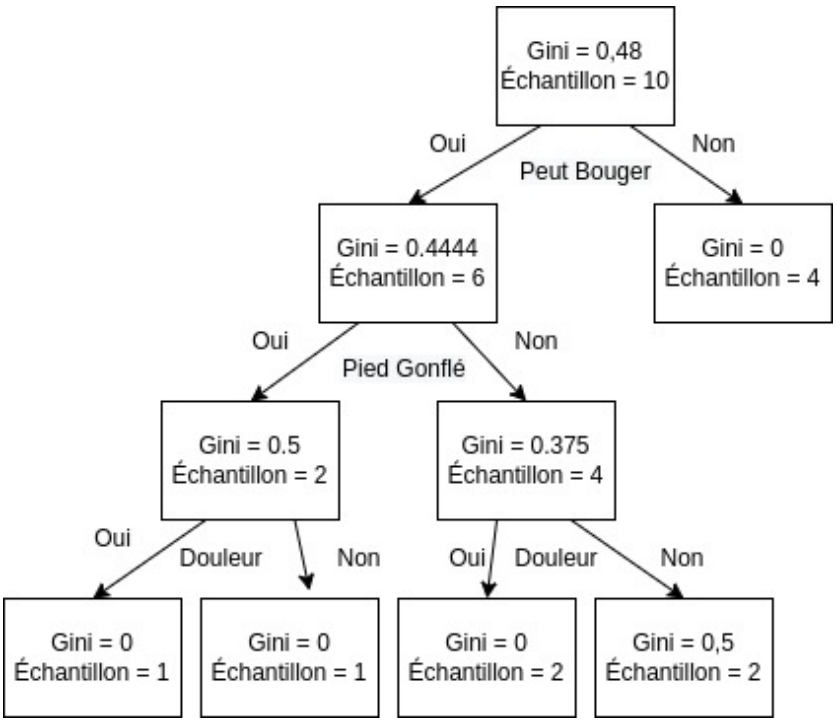


FIGURE 2.14 – Exemple d'arbre de décisions

Quand un échantillon est à traiter, il va parcourir l'arbre jusqu'à ce qu'il termine dans une feuille où il sera assigné à la classe majoritaire. Il existe de nombreux algorithmes d'arbres, dans ce document nous utilisons principalement des arbres de type CART [BFOS17]. En fonction du type d'algorithme utilisé, les principaux paramètres seront : la profondeur maximale de l'arbre, le nombre maximum de feuilles ou encore la taille minimum de l'échantillon pour effectuer une séparation.

2.3.2.3 Forêt d'arbres décisionnels

Une forêt d'arbres décisionnels [Bre01] ou random forest (RF) utilise plusieurs arbres de décisions pour faire une prédiction. Les arbres de décisions

que nous avons utilisés sont de type CART [BFOS17]. Pour la création des arbres de décisions deux techniques sont utilisées. Tout d'abord une stratégie dite de "bagging" : chaque arbre est entraîné sur un ensemble de données de taille équivalente au jeu de données d'apprentissage issu d'un tirage aléatoire avec remise sur le jeu de données d'apprentissage. Chaque arbre est donc entraîné avec des données légèrement différentes. La deuxième technique utilisée intervient au moment de la sélection des caractéristiques à utiliser pour créer deux nouvelles branches. En effet toutes les caractéristiques ne vont pas être utilisées, un sous-ensemble de taille $\sqrt{Q}$ va être sélectionné aléatoirement parmi les Q caractéristiques. Le choix de la meilleure caractéristique et de la meilleure valeur de séparation se fait ensuite à l'aide de l'indice de pureté de Gini (choix de la caractéristique qui minimise celui-ci). Il est à noter qu'il existe un modèle similaire à celui-ci nommé ExtraTrees [BJB19] dans lequel le choix est fait aléatoirement. Une fois tous les arbres entraînés, la décision finale se fait par vote majoritaire.

Si l'on a H arbres de décisions la formule pour la prédiction finale pour un exemple x est :

$$f(x) = \frac{1}{H} \sum_{h=1}^{H} f_h(x) \tag{2.17}$$

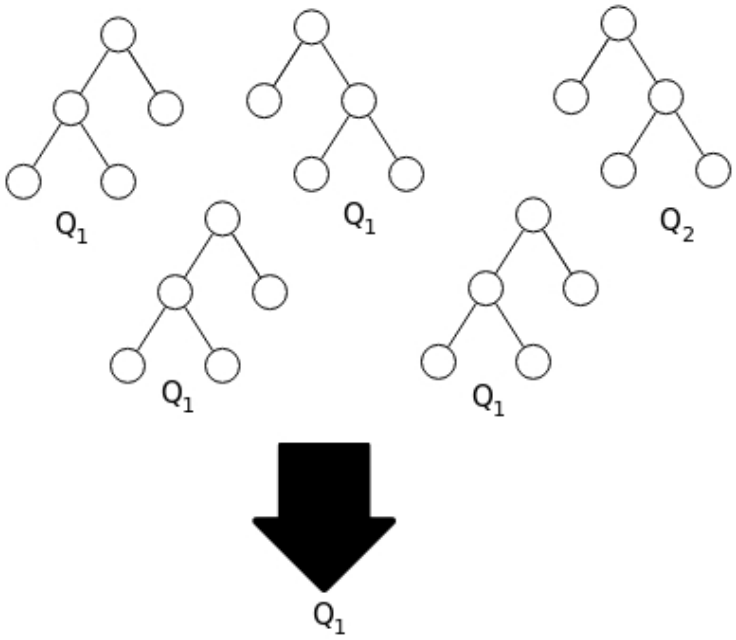


FIGURE 2.15 – Forêt d'arbres décisionnels

Ce classifieur est utilisé dans les toutes les sections du chapitre 3 et dans la section 4.3 du chapitre 4.

2.3.2.4 Boosting

Nous allons ici présenter Adaboost qui signifie adaptative boosting. C'est un algorithme utilisant des arbres pour faire des prédictions. Contrairement à la forêt d'arbres décisionnels où les arbres sont construits de manière parallèle, ici les arbres sont construits de manière séquentielle. Les arbres utilisés sont des classifieurs dits faibles et sont généralement des souches, c'est à dire un noeud avec seulement deux feuilles, entraînés sur un sous-ensemble du jeu de données. Chaque nouveau classifieur est entraîné en prenant en compte les résultats des arbres précédents pour pondérer les exemples. De plus, les arbres ont eux aussi un poids.

Si l'on a n couples (x_n,y_n) dans notre jeu d'apprentissage avec $y \in \{-1, 1\}$ (classification binaire), T classifieur faibles, on associe aux exemples la distribution de probabilité $D_t(x_i)$, celle-ci est initialement uniforme sur les x_i. On note h_t le t-ième classifieur, $\epsilon_t = \frac{1}{2}\sum_{i=1}^{n} D_t(x_i) * |h_t(x_i) - y_i|$ l'erreur associée à ce classifieur et $\alpha_t = \frac{1}{2} * ln(\frac{1-\epsilon_t}{\epsilon_t})$ la confiance que l'on a dans ce classifieur.

L'algorithme fonctionne en plusieurs étapes pour chaque t jusqu'à atteindre T.

- Tirer aléatoirement un échantillon d'exemples à l'aide $D_t(x_i)$, puis entraîner h_t
- Evaluer ϵ_t en utilisant l'ensemble des données
- Calculer α_t
- Mettre à jour $D_t + 1(x_i) = \frac{D_t(x_i)e^{(-\alpha_t y_i h_t(x_i))}}{\sum_{i=1}^{n} D_t(x_i)e^{(-\alpha_t y_i h_t(x_i))}}$ puis retourner à la première étape jusqu'à atteindre T.
- la prédiction finale pour un exemple inconnu se fait enfin avec la formule $F(x) = signe(\sum_{t=1}^{T} \alpha_t h_t(x))$

Les deux premières étapes sont en réalité effectuées pour chaque caractéristique avec une souche afin de déterminer le h_t à utiliser (celui qui aura la plus faible erreur ϵ_t).

Ce classifieur est utilisé dans les sections 3.4 et 3.5.

2.3.2.5 Gradient tree Boosting

Le "gradient tree boosting" est un algorithme utilisant de multiples classifieurs faibles pour faire une prédiction. Les classifieurs faibles sont des arbres de régression avec une profondeur modérée. Les arbres sont entraînés de manière séquentielle. Chaque nouvel arbre est entraîné sur les erreurs (nommées

pseudo-résidus) du modèle global issu de l'étape précédente. Le modèle global est ensuite mis à jour à l'aide des prédictions du nouveau classifieur faible pondéré par le pas d'apprentissage.

Algorithm 1 Gradient tree boosting

input : jeux de donnée (x,y), fonction de coût L, nombre d'itérations M, τ le pas d'apprentissage
initialisation : $F_0(x) = \text{Argmin}_\nu \sum_{i=1}^{n} L(y_i, \nu)$
for $m \leftarrow 1$ **to** M **do**
  for $i \leftarrow 1$ **to** n **do**
    Calculer les pseudo-résidus $r_{i,m} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$
  end
  Entrainer un arbre de décisions sur $\{(x_i, r_{im})\}_{i=1}^{n}$ et créer les régions terminales $R_{j,m}$ pour $j = 1$ à j_m

  for $j \leftarrow 1$ **to** j_m **do**
    calculer $\nu_{j,m} = \text{Argmin}_\nu \sum_{x_i \in R_{j,m}} L(y_i, F_{m-1}(x_i) + \nu)$
  end
  Mettre à jour le modèle : $F_m = F_{m-1} + \tau * \sum_{j=1}^{j_m} (x \in R_j, m)\nu$
end
OUTPUT $F_M(x)$

Dans ce document nous utilisons deux implémentassions différentes de l'algorithme de gradient boosting, Catboost [PGV+18] et lightGBM [KMF+17]. Ces algorithmes sont utilisés dans toutes les sections du chapitre 4. Ce type de classifieur est celui qui, selon la littérature, offre les meilleurs résultats sur les données tabulaires [GOV22].

2.3.2.6 Régression logistique

La régression logistique, contrairement à ce que son nom indique, est un classifieur (on présentera ici la version binaire). Si l'on note β le vecteur de paramètres du modèle et x le vecteur de taille k représentant les variables explicatives :

$$P(1|x) = \frac{exp(\beta_0 x_0 + \beta_1 x_1 + ... + \beta_k x_k)}{1 + exp(\beta_0 x_0 + \beta_1 x_1 + ... + \beta_k x_k)} \tag{2.18}$$

Le but va ainsi être de déterminer les coefficients β. Pour ce faire, on va maximiser la fonction de Log-vraisemblance suivant pour un échantillon de taille n :

$$l(\beta) = \sum_{i=1}^{n} [y_i log(P_i) + (1 - y_i) log(1 - P_i)] \tag{2.19}$$

La maximisation se fait généralement à l'aide d'une descente de gradient [Bur19], la prédiction finale est ensuite prise en seuillant $P(1|x)$.

Cet algorithme est utilisé comme meta-classifieur dans la section 3.5.

2.3.2.7 Réseau de neurones

Un réseau de neurones est une fonction composée de la forme $y = f_{nn}(x)$. Le réseau de neurones se compose de couches de neurones (représentant des valeurs numériques), chaque neurone est relié aux autres par des poids. Le réseau est dit totalement connecté si tous les neurones de la couche précédente sont reliés à tous ceux de la couche suivante.

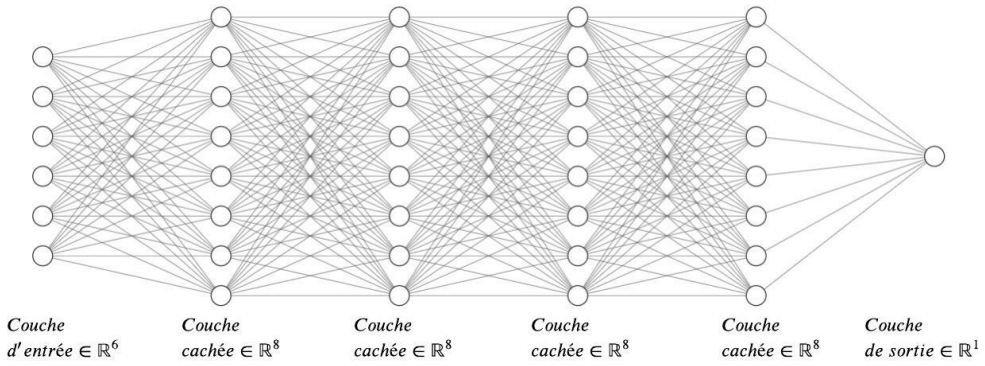


FIGURE 2.16 – Structure d'un réseau de neurones totalement connectés (les lignes représentent les poids) pour la classification binaire

Le réseau est dit à propagation avant quand la valeur contenue dans les neurones d'une couche est calculée à l'aide des poids, des valeurs des neurones de la couche précédente et d'une fonction d'activation. Si l'on note l l'indice de la couche (allant de 1 au nombre total de couches), g_l la fonction d'activation, W_l la matrice des poids, b_l le vecteur de biais on calcule et a_l on a

$$a_{l+1} = g_l(W_l a_l + b_l) \tag{2.20}$$

Dans le cas binaire la décision finale est prise à l'aide de la fonction logistique ($f(x) = \frac{1}{1+exp(-x)}$) dans le cas binaire et de la fonction softmax dans le cas multiclasse. Pour entraîner le réseau de neurones on va utiliser une fonction de coût qui va être utilisée pour comparer les résultats du réseau à la vérité de terrain. La fonction de coût est minimisée (en mettant à jour les poids du réseau) à l'aide d'une descente de gradient et de l'algorithme de rétro-propagation du gradient [Roj96].

Ce type d'algorithme est utilisé dans les sections 3.4 et 3.5. Il est à noter que l'architecture présentée ici (perceptron multicouche) n'est utilisée que dans la section 3.4. En effet, dans la 3.5 nous avons utilisé une autre architecture nommée google Tabnet [AP21]. Cette autre architecture spécialisée

pour les données tabulaires, elle a pour objectif de concurrencer les algorithmes de gradient boosting (qui dominent le domaine). Cette architecture neuronale différente utilise le principe d'attention. L'avantage de ce modèle est de ne nécessiter aucun pré-traitement sur les données et de n'avoir qu'un faible nombre d'hyperparamètre.

2.3.2.8 Stacking

Les méthodes de combinaisons de modèles ont pour objectif de combiner plusieurs modèles forts entre eux pour créer un nouveau modèle que l'on espère meilleur. Chaque modèle fait une prédiction et un vote à majorité a lieu pour choisir la prédiction finale. Dans le cas du stacking un méta-modèle est construit pour effectuer les prédictions finales à partir des prédictions des autres classifieurs. Le meta classifieur que nous utilisons est une régression logistique, celui-ci est entraîné à l'aide des prédictions des autres classifieurs.

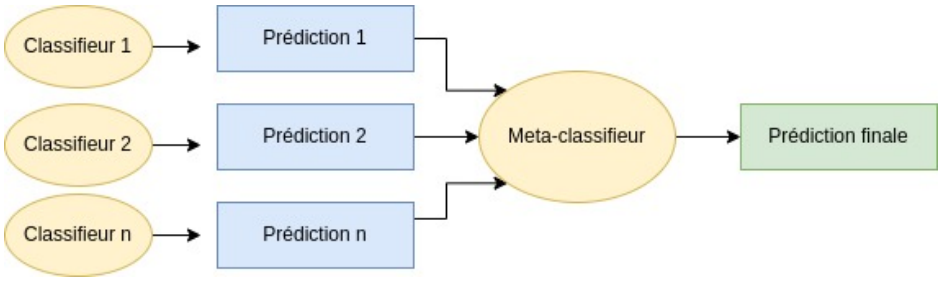


FIGURE 2.17 – Stacking de classifieurs

Cette méthode est utilisée dans la section 3.5.

2.3.2.9 Calibration

Les algorithmes d'apprentissage automatique renvoient la probabilité qu'un exemple appartienne à une classe sachant le vecteur de caractéristiques de cet exemple. Cependant cette probabilité ne correspond pas à la vraie probabilité à posteriori, elle est biaisée par le modèle. Chaque type de modèle d'apprentissage a un biais différent. Une méthode pour corriger ces probabilités est la régression isotonique.

Si on note $\overline{y}_i$ la prédiction faite par le modèle, y_i la valeur réelle , f une fonction croissante monotone. L'hypothèse de la méthode est que :

$$y_i = f(\overline{y}_i) + \epsilon_i \tag{2.21}$$

Ainsi pour un jeu de données $(y_i,\overline{y}_i)$ on va chercher à déterminer $\widehat{f}$:

$$\widehat{f} = argmin_{\xi} \sum_{i=1}^{n} (y_i - \xi(\overline{y}_i))^2 \tag{2.22}$$

Résoudre ce problème se fait à l'aide de l'algorithme PAV [ABE+55] (pair-adjacent violators). Il est à noter que les données utilisées pour résoudre ce problème ne doivent pas être les mêmes que celles ayant été utilisées pour l'entraînement du modèle d'apprentissage automatique afin de ne pas introduire de biais.

2.3.2.10 Plongement de mots

Dans tous les modèles que nous avons évoqués jusqu'à présent les caractéristiques sont supposées numériques. En effet, les chaînes de caractères ne peuvent être utilisées directement pour effecter un apprentissage. Les techniques de plongement de mots vont permettre de transformer un mot en une valeur numérique ou un vecteur qui pourra ensuite être utilisé comme caractéristique.

Quand le nombre de mots à encoder est très limité on peut utiliser des méthodes très simples comme un encodage binaire ou le "one hot encoding". Cependant, lorsque l'on traite de gros volumes de textes il existe des méthodes plus sophistiquées qui vont permettre de prendre en compte le contexte ainsi que la similarité sémantique et syntaxique des mots lors de l'encodage. De tels modèles nécessitent lors de leur entraînement de grand corpus de textes et sont très coûteux à entraîner, on a donc généralement utilisé un modèle pré-entraîné qui nous donnera la valeur vectorisée des mots qu'on lui présente. Le modèle de plongement de mots évoqué dans ce document est GLOVE [PSM14], le modèle est basé sur la matrice de co-occurrence des mots dans l'ensemble du corpus.

2.3.3 Les modèles non-supervisés que nous avons utilisés

2.3.3.1 Analyse en composante principale

L'analyse en composante principale (ACP) est une méthode linéaire qui permet de convertir un ensemble de variables possiblement corrélées en un ensemble de variables non corrélées [F.R01] appelées composantes principales. Le premier axe dans le nouveau système de coordonnées se trouve dans la direction de la plus haute variance dans les données. Le deuxième axe est perpendiculaire au premier et se situe dans la direction de la deuxième plus haute variance.

Algorithm 2 Analyse en composantes principales [Gro09]

INPUT : Jeu de données X

for *i=1 to n caractéristique* **do**
Centrer les données
Calculer l'écart type
Calculer la matrice de covariance C
Calculer les valeurs propres et les vecteurs propres de la matrice de covariance $V^1CV = D$
Réarranger les valeurs propres et les vecteurs propres
Ordonner par ordre décroissant les colonnes de la matrice des vecteurs propres V et la matrice des valeurs propres D
Calculer l'énergie cumulée de chaque vecteur propre :
g[m] = $\sum_{q=1}^{m}$D[p,q] for p=q et m=1,..,M
OUTPUT le sous ensemble de vecteurs propres

2.3.3.2 Auto-Encodeur

Un auto-encodeur est un réseau de propagation avant, composé de couches cachées totalement connectées. Le réseau se décompose en deux parties, l'encodeur et le décodeur. L'objectif d'un auto-encodeur est d'apprendre à reproduire au mieux le signal qui lui est appliqué en entrée.

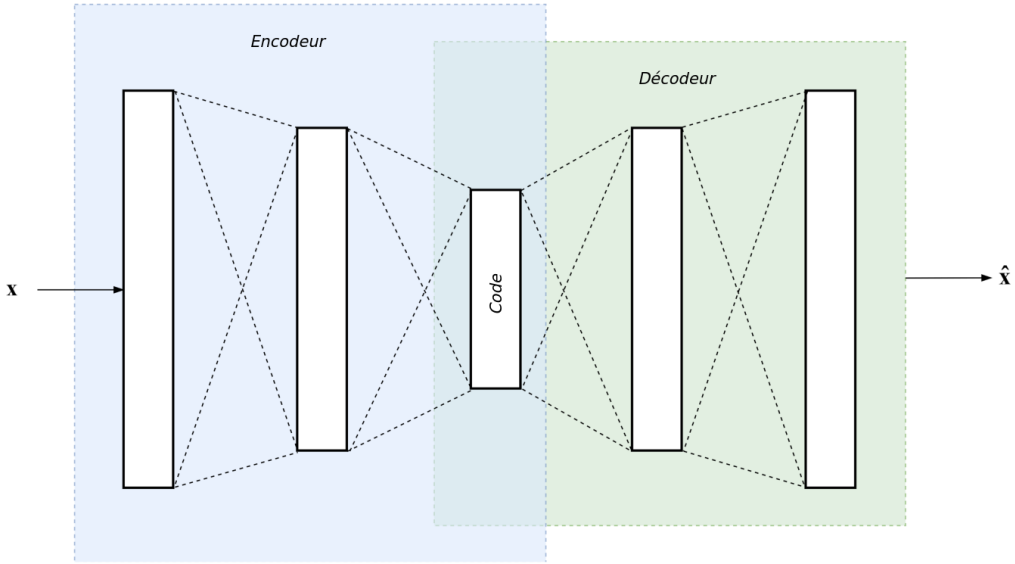


FIGURE 2.18 – Auto-encodeur [Bur19]

Durant ce processus les vecteurs d'entrée de taille $x \in \mathcal{R}^d$ passe par des couches de neurones de plus en plus petites (c'est l'encodeur) jusqu'à ce que le vecteur obtenu soit de taille $x \in \mathcal{R}^p$ tel que $p < d$ ce vecteur est appelé le code. Le vecteur r passe ensuite par des couches de plus en plus grandes (symétriques aux précédentes) et reconstruisent un vecteur $\widehat{x}$

de même dimension que x. Dans le cas le plus où l'on a qu'une seule couche cachée on peut écrire :

$$Encodeur\ : r = f(Wx + b) \tag{2.23}$$

$$Décodeur\ : \widehat{x} = g(W'r + b') \tag{2.24}$$

Avec W et W' des matrices de poids de taille $p \times d$, b et b' des vecteurs de biais et f et g des fonctions d'activation.

L'erreur de reconstruction est généralement mesurée à l'aide de la méthode des moindres carrés ordinaires si les entrées sont des vecteurs réels. Si on a N exemples :

$$Erreur = \frac{1}{N}\sum_{n=1}^{N} \|x^{(n)} - g(W'f(Wx^{(n)} + b) + b'\|^2 \tag{2.25}$$

Quand les vecteurs prennent des valeur entre 0 et 1 on préférera utiliser l'entropie croisée.

Cet algorithme est utilisé dans la section 4.3.

2.3.3.3 Stochastic Neighbor Embedding

Soit $X = \{x_1, \ldots, x_N\}$ un jeu de données avec N individus dans un espace de grande dimension doté d'une distance $d(\cdot, \cdot)$. De manière générale, les méthodes de réduction dimensionnelle pour la visualisation des données vont chercher à associer à chaque individu de X de l'espace de haute dimension un individu $Y = \{y_1, \ldots, y_N\}$ dans un espace bidimensionnel ou tridimensionnel tout en préservant certains aspects des relations topologiques présentes au sein de X. Stochastic Neighbor Embedding (SNE) commence par convertir les distances disponibles $d(x_i, x_j)$ dans l'espace de grande dimension considéré (la distance euclidienne) $\|x_i - x_j\|$) en une probabilité $p_{j|i}$ qui représente la similarité entre x_j x_i. Pour calculer cette probabilité nous utilisons le noyau gaussien suivant :

$$p_{j|i} = \frac{\exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma_i^2}\right)}{\sum_{k=1,k\neq i}^{N} \exp\left(-\frac{\|x_i - x_k\|^2}{2\sigma_i^2}\right)},\ \text{et}\ p_{i|i} = 0, \tag{2.26}$$

où σ_i^2 est la variance du noyau gaussien, qui dépend de la position de l'individu x_i. Afin de déterminer la valeur de σ_i nous utilisons, comme paramètre

de perplexité $2^{-\sum_j p_{j|i} \log_2 p_{j|i}}$ qui est égal à une valeur définie par l'utilisateur u. Ce calcul peut être effectué par recherche dichotomique [HR03] ou par des méthodes robustes de recherche de zéros [VCPn13].

De la même manière, dans l'espace de petite dimension, on calcule la probabilité $q_{j|i}$ pour la distance euclidienne $\|y_i - y_j\|$ en utilisant le noyau gaussien normalisé suivant :

$$q_{j|i} = \frac{\exp\left(-\frac{\|y_i - y_j\|^2}{2}\right)}{\sum_{k=1,k\neq i}^{N} \exp\left(-\frac{\|y_i - y_k\|^2}{2}\right)}, \text{ et } q_{i|i} = 0. \tag{2.27}$$

Remarquons que dans l'espace de petite dimension nous n'avons pas l'adaptabilité du noyau par rapport à la variance. Pour déterminer les coordonnées de y_i, on utilise les distributions P_i et Q_i correspondant respectivement aux distributions de probabilités dans l'espace de grande et de petite dimension. Les coordonnées de y_i sont obtenues par la minimisation de la divergence de Kullback-Leibler entre P_i et Q_i :

$$C_{SNE}(Y) = \sum_{i=1}^{N} KL(P_i \| Q_i) = \sum_{i=1}^{N} \sum_{j=1,j\neq i}^{N} p_{j|i} \log \frac{p_{j|i}}{q_{j|i}} \tag{2.28}$$

Pour la minimisation de (2.28) nous utilisons le gradient suivant :

$$\frac{\partial C_{SNE}(Y)}{\partial y_i} = 2 \sum_{j\neq i} (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j) \tag{2.29}$$

Remarquons que dans l'espace de basse dimension $q_{j|i} \neq q_{i|j}$ tandis que dans l'espace de grande dimension $p_{j|i} \neq p_{i|j}$, donc ces probabilités ne sont pas symétriques. On peut aussi remarquer le "crowding problem", c'est-à-dire même pour de petites valeurs de $p_{j|i}$ et $p_{i|j}$, on veut que dans nos visualisations les points y_i and y_j soient bien séparés dans l'espace de petite dimension. Bien que des stratégies existent pour compenser ce problème [HR03], les désavantages de cette méthode ont mené à la création d'une stratégie différente nommée t-SNE.

2.3.3.4 t-Distributed Stochastic Neighbor Embedding

Bien qu'ayant des points communs, les méthodes SNE et t-SNE ont deux grandes différences. Premièrement, la t-SNE s'appuie sur des probabilités symétrisées $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$. Cette symétrisation rend le gradient de la fonction de coût plus simple à optimiser. Ensuite, avec la méthode t-SNE, pour définir la similarité entre les points y_i et y_j dans l'espace de petite dimension, nous utilisons les probabilités q_{ij} qui sont définies dans la t-SNE comme suivant une loi de student avec 1 degré de liberté et non une loi normale. Il en résulte

la formule suivante :

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k,l(k\neq l)} (1 + \|y_k - y_l\|^2)^{-1}}, \text{ et } q_{i|i} = 0. \tag{2.30}$$

Avec la fonction de coût associé :

$$C_{tSNE}(Y) = KL(P\|Q) = \sum_{i=1}^{N} \sum_{j=1,j\neq i}^{N} p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}, \tag{2.31}$$

Ainsi que le gradient donné par :

$$\frac{\partial C_{tSNE}(Y)}{\partial y_i} = 4 \sum_{j\neq i} (p_{j|i} - q_{j|i})(1 + \|y_i - y_j\|^2)^{-1}(y_i - y_j). \tag{2.32}$$

Rappelons que la loi de student a une queue plus épaisse que la loi normale. Ainsi en utilisant la loi de student pour modéliser les petites valeurs de p_{ij}, la distance entre y_i et y_j doit être grande. C'est une mécanique utilisée par la t-SNE pour compenser le "crowding problem".

Afin d'illustrer les algorithmes précédents nous les avons tous utilisés sur le jeu de données Har [AGO+13], composé de 561 dimensions et 6 classes, les résultats sont présentés dans les figures 2.19, 2.20 et 2.21

Cette méthode est utilisée dans les sections 3.4 et 3.5.

2.4 Sélection de caractéristiques

La sélection de caractéristiques est un domaine de recherche ayant pour but de sélectionner à partir d'un ensemble de caractéristiques de taille N un sous-ensemble de taille k $<$ N qui maximisera une certaine fonction d'objectif, le problème est illustré dans la figure 2.22. Cette tâche peut être un problème d'optimisation mono ou multi objectifs en fonction de la fonction d'objectif utilisée. On s'intéressera ici à la version mono-objectif où la fonction à maximiser est le taux de bonne reconnaissance calculé sur plusieurs validations croisées. Bien qu'évaluer l'ensemble des possibilités soit simple quand N est petit, il devient impossible de vérifier toutes les solutions en un temps raisonnable quand N grandit, l'espace de recherche étant de taille 2^N.

Une méthode pour réduire la complexité du problème consiste à fixer k, ainsi l'ensemble des possibilités n'est plus que de taille : $\frac{n!}{k!*(n-k)!}$. Cependant cet espace reste très vaste et l'explorer entièrement n'est que rarement possible. La seule possibilité va donc être d'avoir recours à des algorithmes qui vont tenter d'identifier de bonnes solutions sans explorer l'ensemble de l'espace de recherche.

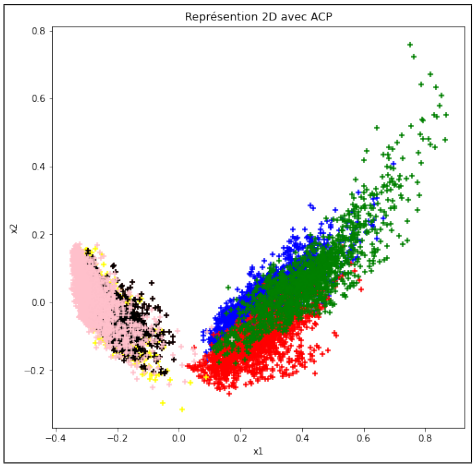


FIGURE 2.19 – Réduction en deux dimensions ACP

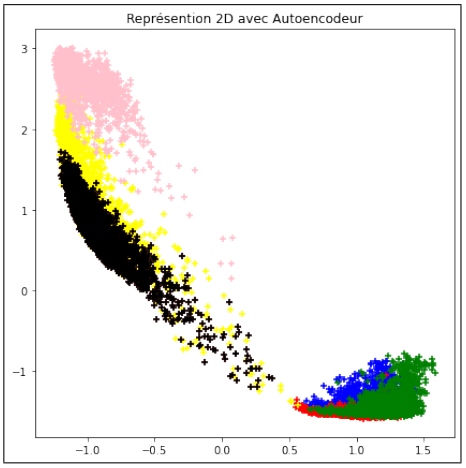


FIGURE 2.20 – Réduction en deux dimensions auto-encodeur

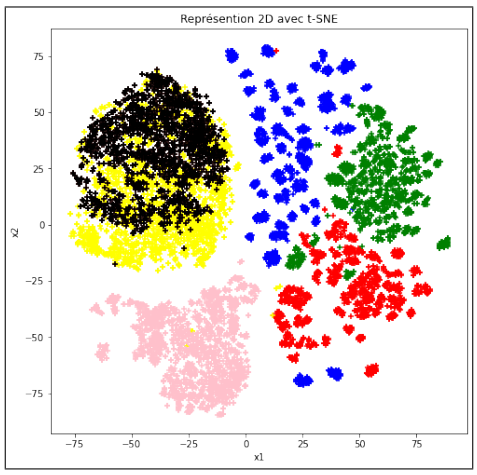


FIGURE 2.21 – Réduction en deux dimensions t-SNE

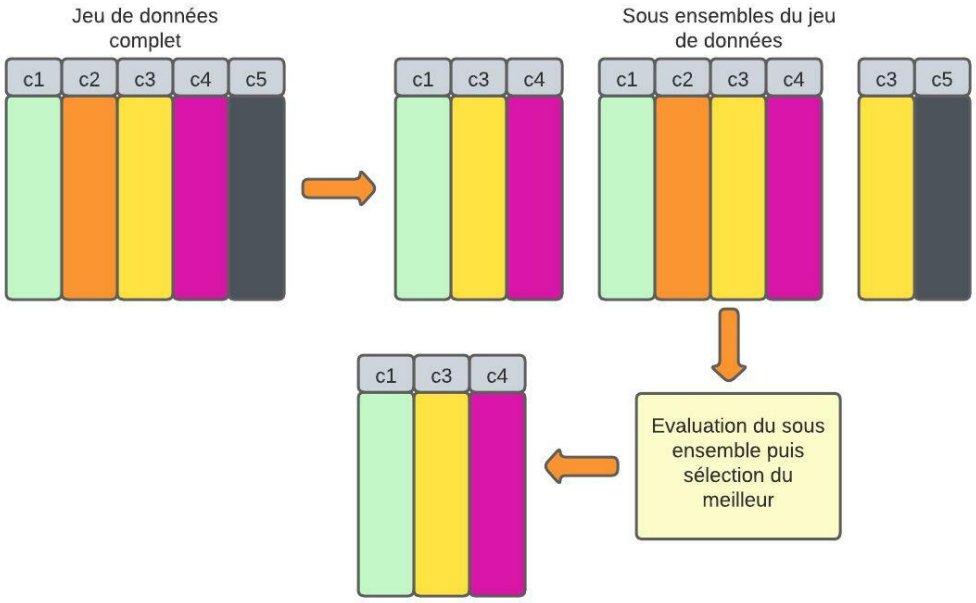


FIGURE 2.22 – La sélection de caractéristiques

Le domaine se compose de trois types de méthodes : Embedded, Wrappers et filters methods [LM12].

Les méthodes filters ne nécessitent pas l'utilisation d'un classifieur, elles sont considérées comme plus rapides et généralistes cependant elles ont tendance à conserver un grand nombre de caractéristiques. Des exemples de ce type d'algorithmes sont : Correlation-based Feature Selection, Information Gain, ReliefF [Hal00] ou encore des méthodes s'appuyant sur la théorie des ensembles approximatifs [QB00].

Les méthodes Wrappers et Embedded nécessitent l'usage d'un classifieur pour sélectionner les meilleures caractéristiques.

Les méthodes Embedded utilisent des classifieurs spécifiques ayant les propriétés requises et la sélection des caractéristiques est intégrée processus d'apprentissage. Les méthodes wrappers s'adaptent à tous les classifieurs et utilisent les résultats de l'apprentissage pour sélectionner les caractéristiques. Un algorithme classique utilisant une méthode de type Embedded est LASSO [Tib96]. La méthode la plus classique de type wrapper est la sélection séquentielle de caractéristiques [ROvdS11] cependant cette méthode est concurrencée par un grand nombre d'algorithmes basés sur des méta-heuristiques[AAGM21]. Il existe aussi l'algorithme Recursive feature Elimination [GWBV02] qui est à l'intersection des deux types de méthodes.

Parmis les méthodes wrappers utilisant des algorithmes basés méta -heuristiques, les plus traditionnelles utilisent des algorithmes évolutionnistes [AFH22], des algorithmes génétiques [HKW01] ou des essaims particulaires [TXZ14].

De nombreux algorithmes sont bio-insipirées et se base sur le comportement d'animaux comme par exemple : la technique de chasse des loups [EZH16], la méthode de chasse des buses de harris [THM+19] ou encore la méthode de chasse des libellules [HMAB+20].

2.4.1 Sélection séquentielle de caractéristiques

Cette méthode, comme son nom l'indique, sélectionne les caractéristiques de manière séquentielle [ROvdS11]. Dans la méthode la plus simple dite "avant" si on désire conserver P caractéristiques, l'algorithme va entraîner un classifieur pour chaque caractéristique puis conserver la meilleure. La même opération va ensuite être recommencée en testant tous les duos de caractéristiques contenant celles sélectionnées à l'étape précédente. L'ensemble de caractéristiques sélectionné va ainsi croître jusqu'à atteindre la taille P souhaitée. Une méthode similaire dite "arrière" démarre en utilisant l'ensemble des caractéristiques et retire à chaque itération la pire caractéristique [Mus17]. Il existe un grand nombre de variations combinant ces deux méthodes afin d'essayer d'obtenir de meilleurs résultats [VC21].

2.4.2 Algorithmes génétiques

Les algorithmes génétiques [Ree10] (AG) sont un sous-ensemble des algorithmes de méta-heuristique ils sont inspirés du fonctionnement de la génétique. L'objectif de ces algorithmes est de déterminer les meilleurs paramètres pour optimiser une fonction d'évaluation.

Dans le cadre de la sélection de caractéristiques les paramètres que l'on va chercher à optimiser sont les caractéristiques à utiliser lors de l'apprentissage. Ces caractéristiques vont être encodés de manière binaire dans un vecteur d'une longueur égale au nombre de caractéristiques total, un 1 caractérisant l'utilisation de la caractéristique lors de l'apprentissage, un 0 son absence.

La fonction à optimiser est dans le cas le plus simple une métrique d'évaluation de l'apprentissage comme le taux de bonne reconnaissance ou le f1-score calculé en moyennant les résultats obtenus sur plusieurs plis d'une validation croisée. La fonction peut aussi être plus complexe et issue de la sclarisation d'un problème multi-objectifs.

Le fonctionnement général de ce type d'algorithme est le suivant. Une population initiale est générée (aléatoirement), chaque individu qui compose cette population contient un ensemble de paramètres appelé chromosome (vecteur binaire de paramètres). Chaque paramètre dans un chromosome est appelé gène.

Ce type d'algorithme s'appuie sur deux mécanismes : la mutation et le croisement. La mutation modifie aléatoirement des gènes dans certains individus (le nombre d'individus mutés étant dépendant d'un paramètre de l'algorithme nommé taux de mutation) et le croisement qui a lieu dans une phase de l'algorithme appelé reproduction (le nombre d'individus croisés dépendant d'un paramètre de l'algorithme appelé taux de croisement). Dans cette phase, deux individus échangent une partie de leurs chromosomes afin de créer deux nouveaux individus appelés "progéniture".

Après chaque génération, tous les individus sont évalués à l'aide d'une fonction d'évaluation et les meilleurs sont conservés pour la génération suivante. Le fonctionnement global est résumé dans la figure 2.23.
Nous avons fait le choix de nous intéresser principalement à l'algorithme génétique GAAM décrit dans la section 4.2, car il permet de réduire très fortement le nombre de caractéristiques à conserver contrairement aux autres algorithmes utilisant des méta-heuristiques. En effet, les autres algorithmes tendent à toujours utiliser entre 40 et 50% du nombre total de caractéristiques même en utilisant une fonction à optimiser prévue pour chercher à minimiser le nombre de caractéristiques à conserver [HMAB+20].

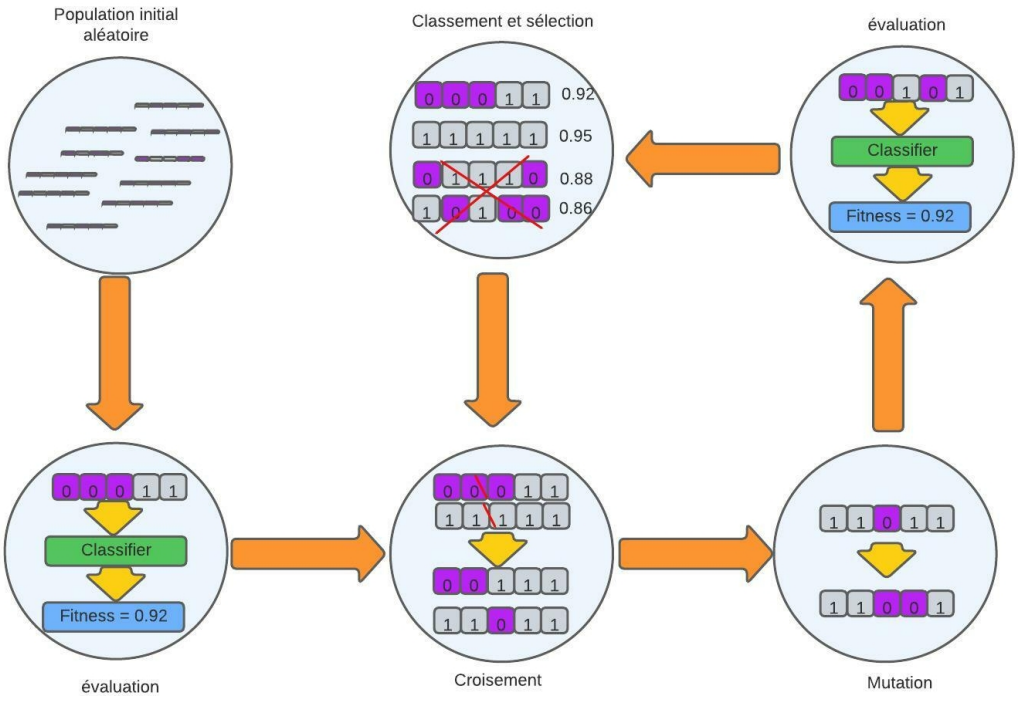


FIGURE 2.23 – Fonctionnement d'un algorithme génétique pour la sélection de caractéristiques

2.5 Conclusion du chapitre

Dans ce chapitre nous avons fait une présentation des principales possibilités de profilage des données mono-colonnes, multicolonnes, mono et multilignes, nous avons aussi effectué un tour d'horizon des applications du profilage des données. Cette présentation permet de fixer le cadre dans lequel le reste du document se déroulera.
De plus nous avons introduit les concepts et algorithmes d'apprentissage automatique qui seront évoqués tout au long de ce document. Nous allons maintenant nous intéresser à la mise en pratique de certains de ces algorithmes pour profiler des données.

CHAPITRE 3

VECTEURS DE MÉTADONNÉES POUR LE PROFILAGE DES DONNÉES

3.1 Introduction 64
3.2 Création du vecteur de caractéristiques 64
3.3 La détection des types sémantiques 68
3.3.1 Introduction 68
3.3.2 Apprentissage pour la détection de types sémantiques . 69
3.3.3 Génération des données 70
3.3.4 Extractions des caractéristiques 71
3.3.5 Modèle d'apprentissage 71
3.3.6 Protocole expérimental 72
3.3.7 Conclusion 79
3.4 Détection des Colonnes quasi-dupliquées 79
3.4.1 Introduction 79
3.4.2 Description générale 80
3.4.3 Description des caractéristiques 81
3.4.4 Principe des algorithmes 81
3.4.5 Généralisation 83
3.4.6 Expériences 84
3.4.7 Résultats 91
3.4.8 Conclusion 92
3.5 La détection des jeux de données quasi-dupliquées 93
3.5.1 Introduction 93
3.5.2 Description du problème 93
3.5.3 Caractéristiques 93
3.5.4 Algorithmes 94
3.5.5 Expériences 97
3.5.5.1 Influence du paramètre ζ 97
3.5.5.2 Influence du nombre de colonnes 97
3.5.5.3 Évaluation de plusieurs classifieurs 97
3.5.5.4 Optimisations 98
3.5.6 Résultats 99
3.5.6.1 Influence du paramètre ζ 99
3.5.6.2 Influence du nombre de colonnes 99
3.5.6.3 Évaluation de plusieurs classifieurs 99
3.5.6.4 Optimisations 105
3.5.7 Conclusion 106
3.6 Conclusion du chapitre 107

3.1 Introduction

Ce chapitre présente les résultats de nos travaux dans le domaine du profilage des données. Ils se découpent en trois sous-parties correspondant à trois thématiques différentes mais ayant toutes un point commun : l'utilisation d'un vecteur de métadonnées dites "simples" afin d'effectuer des actions de profilage plus complexes. Ce vecteur de métadonnées est une version modifiée de celui utilisé dans Sherlock [HHB+19] et réutilisé dans Sato [ZHS+20]. Initialement utilisé pour l'identification par apprentissage automatique du type sémantique des données nous avons étendu son usage à d'autres problèmes.

3.2 Création du vecteur de caractéristiques

La transformation de colonnes de différentes longueurs en vecteurs de taille fixe est une étape pré-requise à l'usage de la plupart des modèles d'apprentissage automatique. La méthode que nous utilisons est inspirée de celle présentée par Sherlock [HHB+19] mais nous ne réutilisons qu'un sous-ensemble des caractéristiques présentées dans Sherlock. Tout d'abord, nous n'utilisons pas les caractéristiques dépendantes directement de la langue des mots de la colonne, donc les caractéristiques de type "Word embeddings" et "Paragraph vectors" (qui utilisent des méthodes classiques de plongement de mot) ne sont pas réutilisées. De plus, nous transformons les données afin que tous les caractères soient en minuscules.

Les 512 caractéristiques que nous extrayons des colonnes sont divisées en deux types principaux. Les caractéristiques dites générales composées de deux sous-catégories. La première se compose de statistiques générales (22 caractéristiques) sur la colonne : entropie, présence ou absence de valeur nulle, pourcentage de chaque type de valeurs (numérique, textuelle, nulle). La moyenne et l'écart type du nombre de cellules contenant des caractères des types suivants : alphabétiques, numériques et spéciaux ou encore la moyenne et l'écart type du nombre de mots dans chaque cellule. La deuxième sous-catégorie de caractéristiques est créée en utilisant des statistiques (Moy, Min, Max, Var, médiane) sur la longueur des chaînes de caractères dans la colonne.

Le deuxième type de caractéristiques est centré sur les caractères, il consiste en des statistiques calculées sur le compte du nombre d'occurrences de chaque caractère dans chaque cellule. Nous avons choisi de traiter les caractères suivants : {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', 'é', 'è','! ', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.', '/', ' :', ';', '<', '=', '>', '? ', '@', '[', ']', '_', '‘', '{', '|', '}', '~', ´, '\x0c'}. Nous utilisons les caractères de l'alphabet latin car nous ciblons des colonnes utilisant cet alphabet, mais la méthode peut être adaptée à d'autres alphabets en

changeant les caractères. Les statistiques calculées sont : min, max, moyenne, médiane, variance, présence ou absence du caractère et présence ou absence du caractère dans toutes les lignes de la colonne. Cela donne un total de 490 caractéristiques. L'ensemble du processus d'extraction des caractéristiques est résumé dans la figure 3.1.

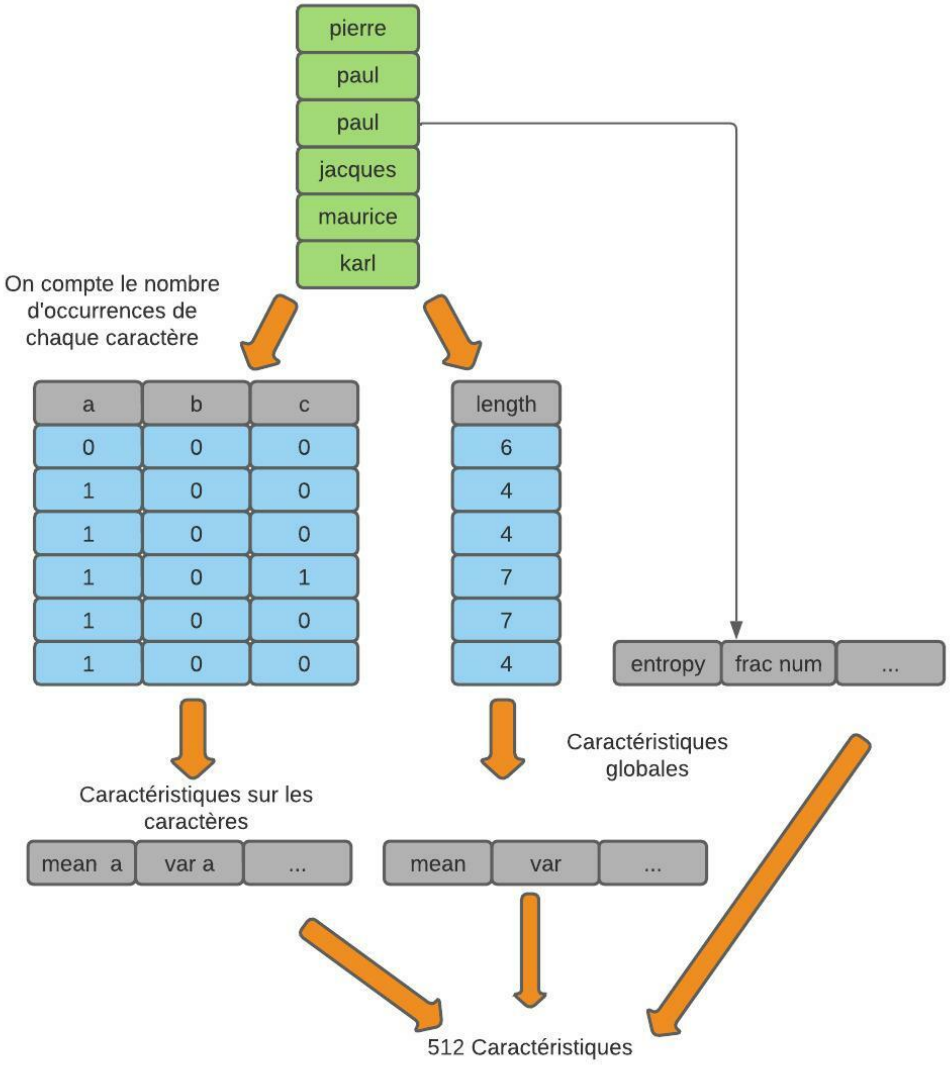


FIGURE 3.1 – Processus de création du vecteur de caractéristiques

Une modification importante est le fait que contrairement à la méthode originale nous n'utilisons pas de caractéristiques qui risquent de croître infiniment avec le nombre de lignes..

Afin d'illustrer la différence entre les deux méthodes d'extraction de caractéristiques, nous utiliserons une colonne de 30000 éléments contenant des titres de films français (FILM-FRENCH). Nous allons ensuite sélectionner

aléatoirement 300 sous-ensembles de tailles {5, 10, 20, 30, 50, 100, 200, 500, 1000, 2000, 5000}. Nous extrairons ensuite de chacun de ces sous-ensembles les caractéristiques utilisant les deux méthodes (en privant Sherlock des caractéristiques de types "Word embeddings" et "Paragraph vectors"), puis nous calculerons la norme euclidienne du vecteur de caractéristiques. La moyenne des normes pour chaque taille d'échantillons pour les deux méthodes est présentée dans les figures 3.2 et 3.3.

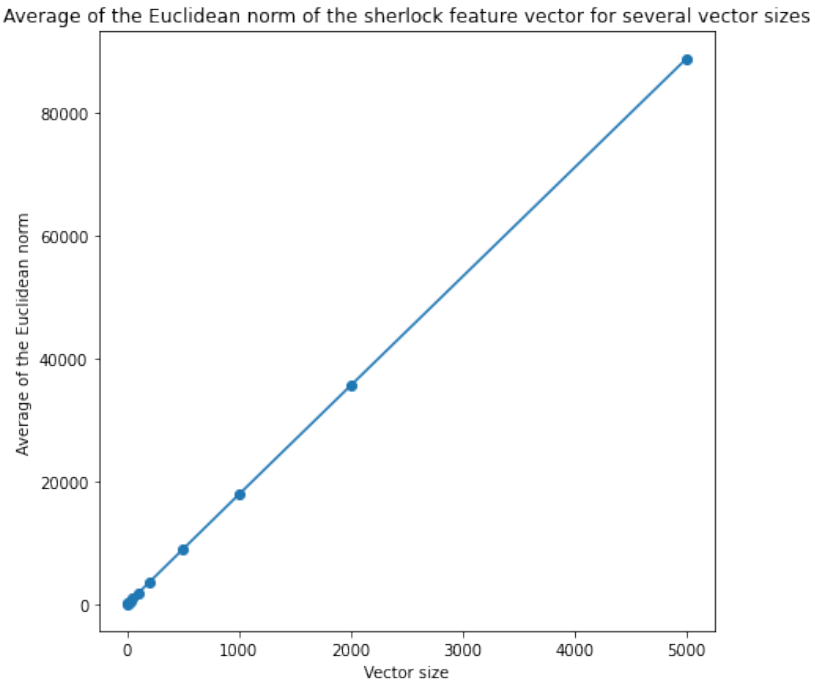


FIGURE 3.2 – Moyenne de la norme euclidienne du vecteur caractéristique utilisant la méthode de Sherlock pour plusieurs longueurs de vecteur (pour une colonne de type sémantique FILM-FRENCH)

Ces représentations nous permettent de constater que notre méthode crée un vecteur de caractéristiques dont les valeurs ne tendent pas vers l'infini avec l'augmentation de la taille de la colonne cible, alors que plusieurs caractéristiques issues de la méthode de Sherlock croissent indéfiniment et sont biaisées par le nombre de lignes.

Le tableau 3.1 représente l'écart-type des résultats de l'expérience précédente. Nous pouvons constater que notre vecteur de caractéristiques a un écart-type élevé lorsque le nombre de lignes dans la colonne est faible.

Un autre point important concernant ces caractéristiques est la vitesse d'extraction. En effet, c'est un facteur limitant. Au-delà de 100 000 lignes, il devient trop long d'extraire les caractéristiques. Le temps d'extraction en fonction du nombre de lignes est représenté dans la figure 3.4 et le tableau 3.2.

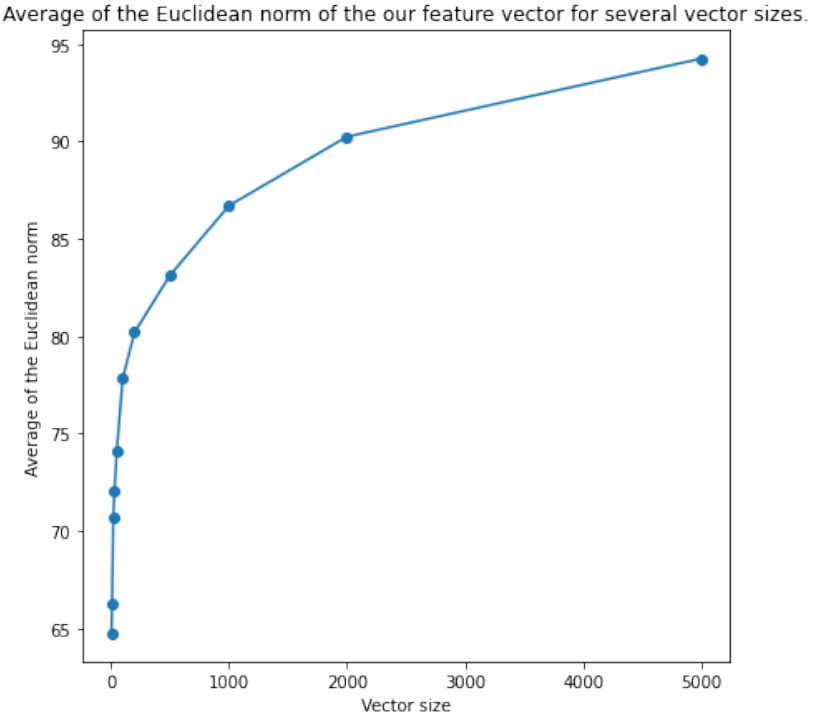


FIGURE 3.3 – Moyenne de la norme euclidienne de notre vecteur de caractéristiques pour plusieurs tailles de vecteur (pour une colonne de type sémantique FILM-FRENCH)

TABLE 3.1 – Écart-type de la norme euclidienne du vecteur caractéristique de Sherlock pour plusieurs tailles de vecteur (pour une colonne de type sémantique FILM-FRENCH) pour les deux méthodes

taille	std Sherlock	std de nos caractéristiques
5	34	36
10	27	25
20	34	22
30	43	16
50	55	15
100	74	11
200	106	8
500	159	6
1000	231	5
2000	374	6
5000	559	8

Nous utilisons une version optimisée de l'implémentation faite dans Sherlock [1] (toujours en python) qui est entre 1.5 et 2 fois plus rapide en fonction du nombre de lignes. Les temps d'exécution présentés sont calculés à partir d'une moyenne de 50 répétitions du même calcul.

1. https ://github.com/mitmedialab/sherlock-project

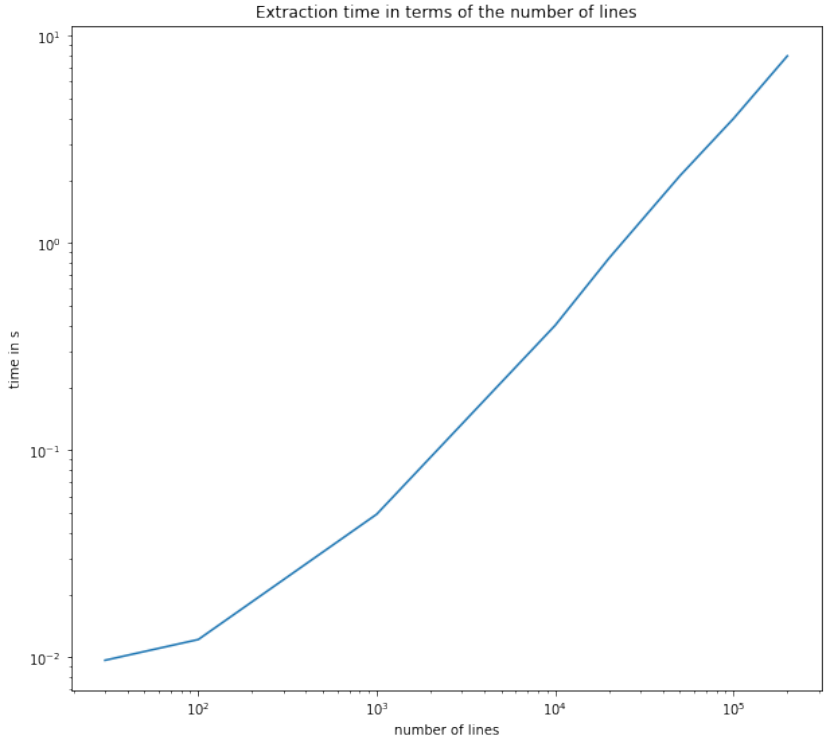


FIGURE 3.4 – Temps d'exécution en fonction du nombre de lignes

TABLE 3.2 – Temps d'exécution en fonction du nombre de lignes

lignes	10000	20000	50000	100000	200000
temps in s	0.40	0.84	2.10	3.98	7.98

Les vecteurs de caractéristiques des sections 3.4 et 3.5, ont des longueurs légèrement différentes. En effet, dans le premier cas le jeu de caractères considéré inclut les majuscules (et certains caractères spéciaux ne sont pas pris en compte), de même dans le second car tous les caractères spéciaux présentés ici ne sont pas pris en compte.

3.3 La détection des types sémantiques

3.3.1 Introduction

Nous allons dans cette section présenter une méthode de détection automatique du type sémantique d'une colonne. Cette méthode est née du désir d'utiliser une méthode similaire à ce qui se fait dans l'état de l'art (utilisant de l'apprentissage automatique) mais qui serait plus accessible et industrialisable. En effet, les méthodes utilisant de l'apprentissage automatique sont nombreuses mais ne sont pas utilisées dans l'industrie qui préfère utiliser des méthodes plus simples [HGG+22].

L'originalité de notre solution est d'offrir une méthode pouvant fonction-

ner sans avoir à récolter de larges volumes de données et permettant de s'adapter à de nouveaux types sémantiques. La méthode proposée est illustrée dans la figure 3.5.

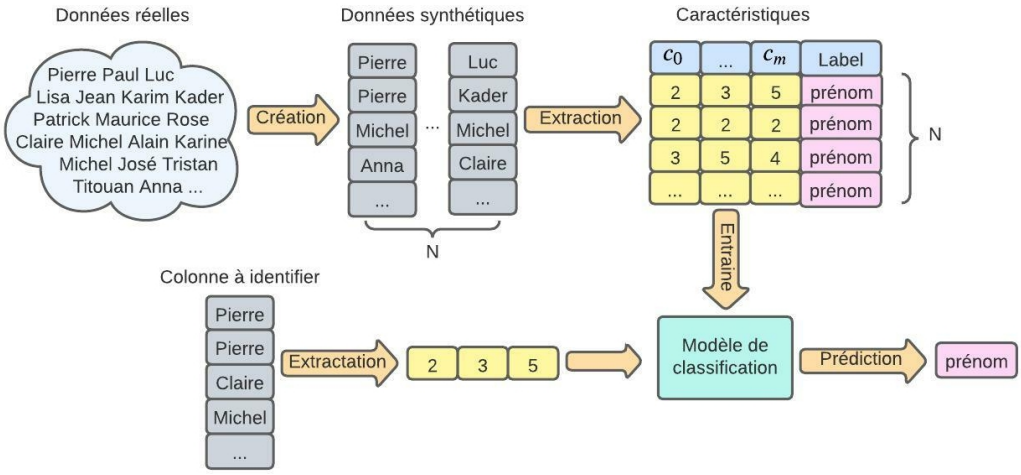


FIGURE 3.5 – Schéma récapitulatif de la méthode proposée

3.3.2 Apprentissage pour la détection de types sémantiques

L'approche proposée dans ce travail diffère de la littérature à plusieurs égards. En effet notre méthode est conçue pour être accessible et réutilisable dans n'importe quel domaine d'activité, indépendamment des catégories sémantiques. De plus, elle est conçue pour pallier à plusieurs défauts de l'algorithme phare du domaine : Sherlock [HHB+19]. Le premier frein à l'usage de Sherlock est le volume de données utilisées. La méthode présentée dans l'article (pour 78 catégories sémantiques) nécessite 60% d'un ensemble de données contenant 686765 colonnes (provenant de VizNet [HGH+19]). Il est donc difficile d'utiliser l'approche Sherlock avec d'autres catégories sémantiques que celles utilisées dans l'article original [HHB+19]. En effet, il est compliqué d'obtenir suffisamment d'exemples pour utiliser sherlock sur des catégories sémantiques personnalisées [DHBS22, LSSB21]. D'autant plus qu'il est démontré dans Sato [ZHS+20] que le taux de bonne reconnaissance de Sherlock dépend fortement du nombre d'exemples présents pour la classe (c'est-à-dire classes non équilibrées). Moins une classe a d'exemples, plus son taux de bonne reconnaissance est faible.
Pour faire face à ce problème, nous avons introduit un algorithme de génération de données artificielles. Contrairement au sur-échantillonnage classique en apprentissage automatique, dans l'algorithme de génération proposé, les données sont créées dans le même espace initial de caractéristiques (variables mixtes). Celui-ci utilise un ensemble contenant le plus grand nombre possible d'éléments du type sémantique des colonnes à créer pour générer à l'aide de la loi de Dirichlet le nombre souhaité de colonnes contenant ces éléments

(3.3.3).
La deuxième limitation est l'utilisation par Sherlock de deux modèles de plongement de mots (word embedding) lors de l'extraction des caractéristiques des colonnes. Cela limite l'utilisation de l'algorithme à une seule langue à la fois. De plus, toutes les langues ne disposent pas de modèles pré-entraînés pour ces algorithmes. Pour éviter ce problème, nous n'avons pas utilisé les caractéristiques provenant de plongement de mots afin que notre méthode soit utilisable dans plusieurs langues.

Le dernier point à discuter est le nombre de lignes dans les colonnes utilisées. Dans Sherlock, le nombre de lignes dans une colonne est une information très importante pour le modèle. Ce nombre est utilisé à la fois directement et indirectement dans le calcul de plusieurs caractéristiques. Cela se traduit par une forte réduction du taux de bonne reconnaissance du modèle lorsqu'il est confronté à des données ne provenant pas de la base de données utilisée dans l'article (où les colonnes n'ont pas la même longueur). Ainsi Khurana et Galhotra montrent que le taux de bonne reconnaissance moyen observé sur 9 jeux de données est de 26,6% pour Sherlock et de 26% pour Sato [KG21, LSSB21]. Afin de résoudre ce problème, nous n'utiliserons ni le nombre de lignes comme caractéristique ni les caractéristiques qui croissent infiniment avec ce nombre.

3.3.3 Génération des données

Dans cette section, nous présentons l'approche proposée pour la génération et l'augmentation des données. Notre méthode de génération tire partie du fait qu'il est facile de générer des vecteurs aléatoires $[x_1 \dots x_K]$ respectant une distribution de Dirichlet de paramètres $\omega_1, \dots, \omega_K$ [NTT11]. La densité de probabilité de la distribution de Dirichlet d'ordre $K \geq 2$ de paramètres $\omega_1, \dots, \omega_K$ est définie comme suit :

$$f(x_1, \dots, x_K; \omega_1, \dots, \omega_K) = \frac{1}{\mathrm{B}(\omega)} \prod_{i=1}^{K} x_i^{\omega_i - 1} \tag{3.1}$$

$$\mathrm{B}(\omega) = \frac{\prod_{i=1}^{K} \Gamma(\omega_i)}{\Gamma\left(\sum_{i=1}^{K} \omega_i\right)}, \qquad \omega = (\omega_1, \dots, \omega_K). \tag{3.2}$$

avec

$$\sum_{i=1}^{K} x_i = 1 \text{ et } x_i \geq 0 \text{ pour tout } i \in \{1, \dots, K\} \tag{3.3}$$

Les vecteurs générés de cette manière ont une somme égale à 1. De plus, nous pouvons modifier la distribution des valeurs dans le vecteur en changeant les paramètres ω. Dans ce qui suit, les paramètres $\omega_1, \dots, \omega_K$ seront désignés par ω car nous utilisons la même valeur pour toutes les dimensions.

Pour générer un vecteur de données de taille l à partir d'un vecteur de

données réelles $[d_1 \dots d_K]$ de taille K (décrivant un type sémantique), nous générons d'abord un vecteur $[x_1 \dots x_K]$. Puis on tire l éléments dans $[d_1 \dots d_K]$ en utilisant comme probabilité de tirage pour chaque d_i la valeur x_i. Plus la valeur de ω est faible, plus il est probable que la distribution des valeurs de $[x_1 \dots x_K]$ soit déséquilibrée. L'algorithme 3 présente une méthode permettant de générer nbr_elem exemples (colonnes) à partir d'un vecteur de données réelles D pour une valeur ω donnée.

Algorithm 3 Data_generation

INPUT : *D :list, nbr_elem :int, ω :int, max_col_size :int, min_col_size :int*
initialisation $x \leftarrow []$
$prb \leftarrow nbr\_elem$ échantillons de dimension longueur(D) à partir d'une distribution de Dirichlet de paramètres ω* "vecteur de uns" de taille longueur(D)

for $i \leftarrow 0$ **to** *nbr_elem - 1* **do**
  $n\_elem \leftarrow$ valeur aléatoire entre min_col_size et max_col_size
  $x[i] \leftarrow n\_elem$ éléments de D choisis suivant la distribution $prb[i]$
end
OUTPUT x

3.3.4 Extractions des caractéristiques

Les caractéristiques utilisées dans cette section correspondent exactement à celles décrites dans la section 3.2.

3.3.5 Modèle d'apprentissage

La méthode proposée utilise deux classifieurs, l'un pour les colonnes contenant de nombreuse lignes nommé "haute", l'autre pour les colonnes en contenant peu nommé "basse". Ce choix a été fait en observant les fortes variations des valeurs dans les vecteurs quand la colonne compte un faible nombre de lignes dans le tableau 3.1. Le seuil entre "haute" et "basse" a été fixé à 30 lignes car la variance des caractéristiques est généralement devenue faible passé ce nombre de lignes.

Pour chaque cas, nous avons essayé deux classifieurs, une RF [Bre01] (en effet il est montré dans Sherlock que cet algorithme donne de bons résultats dans cette tâche) et Catboost [PGV+18] dont les résultats dépassent ceux de la RF pour d'autres tâches utilisant des caractéristiques similaires [CBG+21, CRB+22a, CRB+22b]. Chaque modèle est calibré en utilisant la méthode isotonique sur une validation croisée 5 plis (la probabilité finale est la moyenne des prédictions des 5 classifieurs) [NMC05, ZE02]. Les

deux RF utilisent comme données d'entraînement deux ensembles de données différents. L'un est généré avec des colonnes de petites tailles allant de 5 à 30 lignes et l'autre avec un nombre de lignes allant de 31 à 1000. Chacun de ces jeux de données est composé de 57 classes (une partie des classes est en français et l'autre en anglais, elles sont décrites dans A.2), chaque classe a 21000 exemples. Parmi ces 21000 exemples générés avec l'algorithme 3, 7000 sont générés avec un omega égal à 0.00001 , 7000 avec un omega égal à 0.001 et enfin 7000 avec un omega égal à 1. Nous avons donc au final deux jeux de données de 1197000 exemples chacun.

3.3.6 Protocole expérimental

Nos expériences sont effectuées sur une instance Colab avec un Xeon 2.30GHz 4-core, 25GB de Ram et une tesla P100 (16go). Les paramètres utilisés pour la RF sont : une profondeur maximale de 18 et 200 estimateurs. Les paramètres utilisés pour Catboost sont : 1500 itérations, une profondeur max de 9 et un pas d'apprentissage de 0,04.

Notre première expérience vise à évaluer le taux de bonne reconnaissance de nos deux classifieurs sur des données que nous allons générer à nouveau avec l'algorithme 3. Nous allons donc créer un jeu de données de test pour chacun des deux modèles.
L'un sera généré avec des colonnes dont la taille varie de 5 à 30 lignes, l'autre avec un nombre de lignes compris entre 31 et 9000. Les valeurs ω varient de 10^{-7} à 100 et sont modifiées par puissance de 10. Pour chaque valeur ω, 50 exemples sont générés par classe, nous avons donc un total de 28500 exemples pour chaque jeu de données de test. Les résultats de ces premières expériences sont présentés dans les tableaux 3.3, 3.4 et les figures 3.6 et 3.7.

Dans la quasi totalité des configurations, Catboost surpasse la RF avec un taux de bonne reconnaissance global de 0,9413 contre 0,9310. Ce n'est pas étonnant car Catboost est considéré comme un classifieur plus performant que la RF et offre une plus grande possibilité d'hyperparamétrage. Les deux modèles présentent d'excellents résultats pour des valeurs d'omega supérieures à 10^{-3}. En dessous de cette valeur, les résultats diminuent fortement, malgré des exemples d'entraînement générés avec omega égal à 10^{-5}. Ce phénomène peut-être expliqué par le processus de génération des données. En effet, plus omega est faible, plus on obtient des colonnes qui seront déséquilibrées (ayant une valeur extrêmement sur-représentée par rapport aux autres). Ce déséquilibre ne réduit pas uniformément le taux de bonne reconnaissance de toutes les classes. Les types sémantiques contenant un grand nombre de valeurs différentes ainsi qu'un grand nombre de caractères possibles sont les plus affectés (par exemple les prénoms). Au contraire, ceux qui ont peu de valeurs différentes et peu de caractères différents ne sont pas

affectés (par exemple les genres).

Les tableaux 3.5 et 3.6 présentent les caractéristiques les plus importantes pour les deux modèles et les deux classifieurs, elles sont triées par ordre décroissant. Bien qu'elles ne représentent qu'une petite partie du total des caractéristiques, les statistiques globales sont très représentées. En particulier, les caractéristiques liées à la longueur des chaînes de caractères. En utilisant la taille moyenne, taille maximale et minimale des chaînes, il est facile de séparer de nombreuses classes. Par exemple, les colonnes de genres ou de groupes sanguins auront des chaînes de caractères en moyenne plus courtes, ainsi que des tailles maximales plus faibles que les chaînes des colonnes de numéros de téléphone ou d'URL. Un autre type de caractéristiques important est celui lié au nombre de caractères (ou cellules) contenant des valeurs numériques ou alphabétiques. En effet, grâce à cette information, nous pouvons séparer les classes ne contenant que des caractères alphabétiques, comme les noms et les prénoms, des classes ne contenant que des chiffres, comme les codes postaux ou le code INSEE. Enfin, au niveau des caractéristiques concernant les caractères, nous trouvons des caractères de séparation tels que ' ' ',' ou '-'. Ces caractères ne sont pas présent dans l'ensemble des types sémantiques et offrent donc une bonne capacité de distinction. Par exemple le caractère ' ' soit utilisé pour séparer les classes contenant plusieurs mots comme les noms d'université des classes ne contenant qu'un seul mot comme les noms de continent.

TABLE 3.3 – Taux de bonne reconnaissance des modèles

ω	RF "haute"	Catboost "haute"	RF "basse"	Catboost "basse"
10^{-7}	0.8376	**0.8403**	0.8479	**0.8510**
10^{-6}	0.8593	**0.8807**	0.8706	**0.8734**
10^{-5}	0.8624	**0.9014**	0.8710	**0.8762**
10^{-4}	0.8859	**0.9228**	**0.9062**	0.9020
10^{-3}	0.9579	**0.9762**	0.9444	**0.9548**
10^{-2}	0.9417	**0.9679**	0.9468	**0.9606**
10^{-1}	0.9969	**0.9989**	0.9682	**0.9758**
1	0.9996	**1**	0.9717	**0.9806**
10	**1**	**1**	0.9755	**0.9810**
100	**0.9996**	**0.9996**	0.9775	**0.9831**

Dans l'expérience suivante, nous avons essayé d'aborder un aspect souvent négligé dans l'état de l'art. Le fait que le problème de détection de type sémantique soit dans la plupart des cas un problème de type ensemble ouvert [BB15, SdRRSB13]. Sherlock n'aborde pas directement ce problème mais propose de rejeter les 10% d'exemples ayant les plus faibles probabilités de sortie afin d'améliorer les performances du modèle [HHB+19]. Le

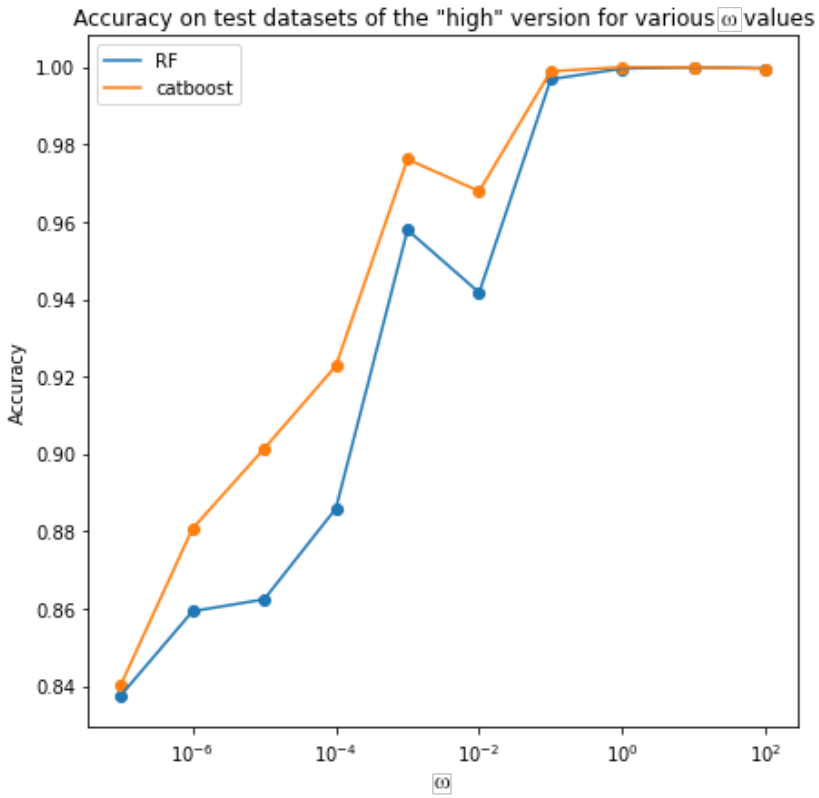


FIGURE 3.6 – Taux de bonne reconnaissance sur les jeux de données de test de la version "haute" pour différentes valeurs de ω.

TABLE 3.4 – F1-score 'macro' des modèles

ω	RF "haute"	Catboost "haute"	Rf "bassse"	Catboost "basse"
10^{-7}	0.8032	**0.8217**	0.8169	**0.8306**
10^{-6}	0.8379	**0.8689**	0.8521	**0.8306**
10^{-5}	0.8515	**0.8978**	0.8620	**0.8696**
10^{-4}	0.8800	**0.9211**	**0.9024**	0.8996
10^{-3}	0.9585	**0.9762**	0.9448	**0.9593**
10^{-2}	0.9397	**0.9684**	0.9466	**0.9607**
10^{-1}	0.9969	**0.9989**	0.9683	**0.9759**
1	0.9996	**1**	0.9716	**0.9806**
10	**1**	**1**	0.9755	**0.9810**
100	**0.9996**	**0.9996**	0.9775	**0.9830**

but de l'approche est de rejeter comme inconnues les colonnes n'appartenant pas à un type sémantique connu. Nous allons réutiliser les modèles décrits dans l'expérience précédente, en ajoutant à nos jeux de données de nouveaux exemples inconnus à rejeter. Ces nouveaux exemples à rejeter proviennent de deux types sémantiques "nom de fromage" et "nom d'association". Pour chacun de ces deux types, 1000 exemples sont générés en suivant le même processus que pour les données de test. La classe prédite par nos modèles

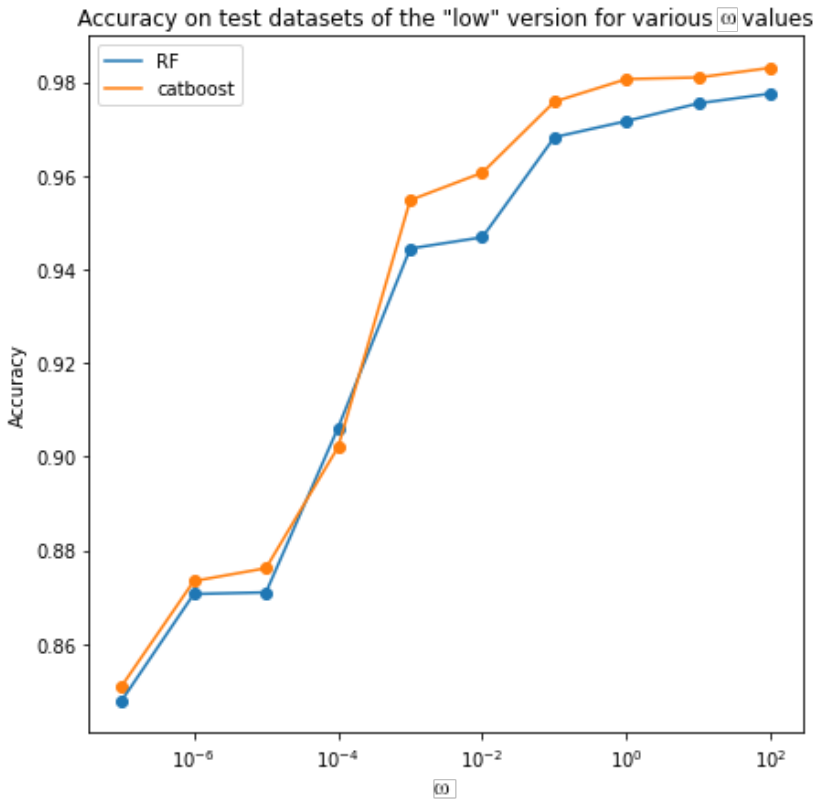


FIGURE 3.7 – Taux de bonne reconnaissance sur les jeux de données de test de la version "basse" pour différentes valeurs de ω.

TABLE 3.5 – Les 10 meilleures caractéristiques pour Catboost, classées par ordre décroissant par leur score d'impureté de Gini.

Catboost "haute"	Catboost "basse"
Moy valeur longueur	Moy valeur longueur
Moy nombre de '-'	Moy nombre de '-'
Moy nombre de 'c'	Min valeur longueur
Max valeur longueur	Moy alphab carac dans cells
Moy nombre de ','	Moy nombre de ','
Moy nombre de 'f'	Moy nombre de 'c'
Moy nombre de 'm'	Max valeur longueur
Min valeur longueur	Moy nombre de 'm'
Fraction cells avec num carac	Moy nombre de 't'
Moy nombre de '0'	Fraction cells avec num carac

est celle dont la probabilité de sortie est la plus élevée. Nos modèles étant calibrés, nous considérerons pouvoir avoir confiance dans la probabilité qu'il retourne. Ainsi en utilisant un seuil minimum pour faire une prédiction, nous pouvons rejeter des exemples.
Dans cette expérience, nous considérerons dans chaque cas que nous avons deux ensembles de données. L'un contenant les données à rejeter (contenant

TABLE 3.6 – Les 10 meilleures caractéristiques pour RF, classées par ordre décroissant par leur score d'impureté de Gini.

RF "haute"	RF "basse"
Moy longueur chaîne	Moy longueur chaîne
Max longueur	Moy alphab carac dans cells
Moy alphab carac dans cells	Max valeur longueur
Mediane longueur chaîne	Mediane longueur chaîne
Min valeur longueur	Min valeur longueur
Moy num carac dans cells	Moy num carac dans cells
Moy de mots dans cells	Moy nombre de 'm'
Moy nombre de ' '	Moy nombre de ' '
Moy nombre de '-'	Moy nombre de 'e'
Moy nombre de 'e'	Moy nombre de '-'

les classes inconnues) et l'autre les données de test (contenant les classes connues) à classer. Nous allons mesurer le taux de bonne reconnaissance des classifieurs sur chacun des deux jeux de données en faisant varier le seuil de rejet. Ainsi, nous considérerons qu'un membre d'une classe connue qui est rejeté est une erreur, et qu'un élément inconnu prédit comme appartenant à une classe connue est une erreur. Les résultats de cette expérience sont présentés dans les tableaux 3.7 et 3.8 et les figures 3.8 et 3.9. Nous avons présenté les résultats uniquement pour la version "haute" de l'algorithme pour les deux classifieurs, les résultats étant similaires pour les versions "basses". Nous pouvons constater que la méthode fonctionne et que le seuil nécessaire pour rejeter la majorité des éléments inconnus est plus bas pour la forêt d'arbres aléatoire qu'avec Catboost malgré la calibration. Il est intéressant de noter que selon l'objectif que l'on se fixe, le classifieur à utiliser n'est pas le même. En effet, si nous voulons trouver le meilleur compromis entre le taux de bonne reconnaissance sur les classes inconnues et celles connues, nous devons choisir la RF avec un seuil de 0.75, qui permet de rejeter 98% des individus inconnus tout en gardant 87% de taux de bonne reconnaissance. Par contre, si nous voulons rejeter le maximum d'individus inconnus, nous devons choisir Catboost avec un seuil de 0.97. Enfin, il est à noter que les performances de rejet dépendent des données à rejeter, plus elles sont similaires aux classes connues plus il est difficile de les rejeter efficacement. Une amélioration possible est de définir un seuil pour chaque classe et non un seuil global pour toutes les classes [SR17].

Notre dernière expérience vise à tester notre modèle avec de nouvelles données et à évaluer ses performances par rapport à un autre algorithme. Les nouvelles données sont constituées de 32 types sémantiques appartenant aux 57 types que nous avons utilisés lors de l'apprentissage. Ces données

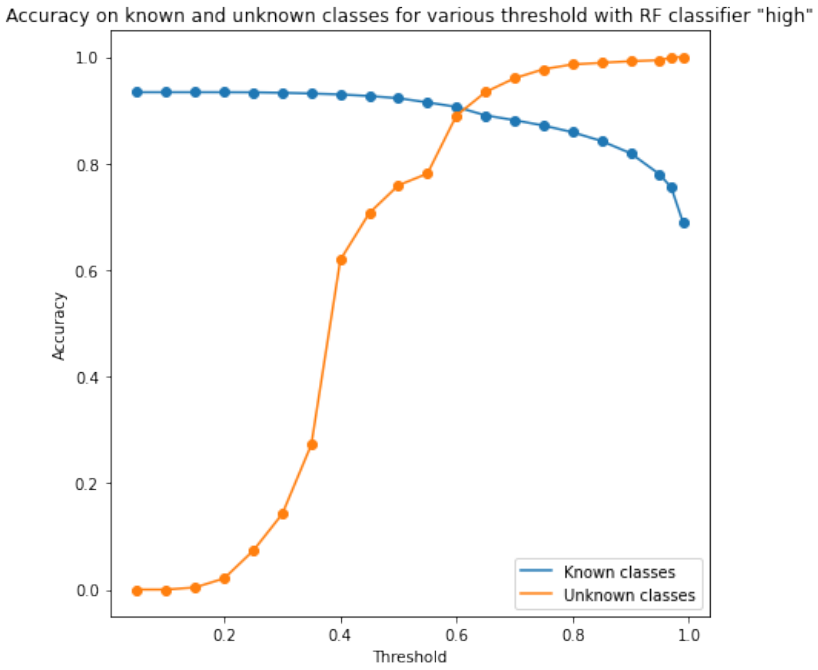


FIGURE 3.8 – Taux de bonne reconnaissance des classes connues et inconnues pour différents seuils avec le classifieur RF 'haute'.

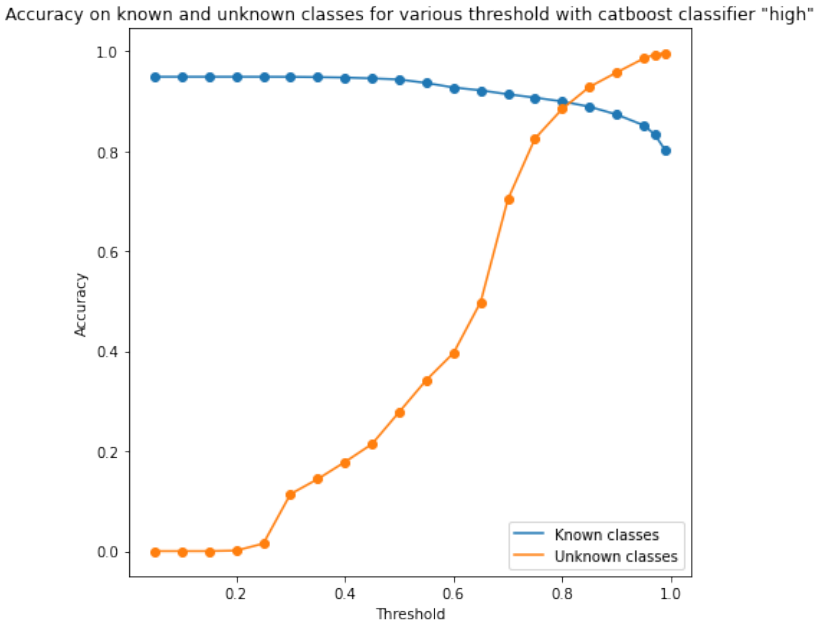


FIGURE 3.9 – Taux de bonne reconnaissance des classes connues et inconnues pour différents seuils avec le classifieur Catboost 'haute'

TABLE 3.7 – Taux de bonne reconnaissance des classes connues et inconnues pour différents seuils avec les classifieurs Catboost RF et Catboost classifier part 1.

seuil	0.35	0.4	0.45	0.5	0.55	0.6	0.65
Catb connu	0.95	0.95	0.95	0.94	0.94	0.93	0.92
Catb inconnu	0.14	0.18	0.21	0.28	0.34	0.40	0.5
RF connu	0.93	0.93	0.93	0.92	0.91	0.91	0.89
RF inconnu	0.27	0.62	0.70	0.76	0.78	0.89	0.93

TABLE 3.8 – Taux de bonne reconnaissance des classes connues et inconnues pour différents seuils avec les classifieurs RF et Catboost classifier part 2.

threshold	0.7	0.75	0.8	0.85	0.9	0.95	0.97
catb known	0.91	0.91	0.90	0.89	**0.87**	0.85	0.83
catb unknown	0.70	0.83	0.88	0.93	**0.96**	0.98	1
RF know	0.88	**0.87**	0.86	0.84	0.82	0.78	0.75
RF unknow	0.96	**0.98**	0.98	0.99	0.99	0.99	1

proviennent d'autres sources de données réelles que celles utilisées pour l'entraînement et n'ont donc pas été générées avec notre méthode.

Pour chaque type sémantique nous avons 20 exemples pour un total de 640 colonnes contenant chacune plusieurs centaines de lignes. L'algorithme que nous utilisons pour la comparaison est un algorithme utilisant des dictionnaires et des expressions régulières.

Ainsi, pour les types sémantiques où cela est possible, un expert humain a défini les expressions régulières. Pour les types sémantiques qui ne peuvent pas être définis à l'aide d'une expression régulière (par exemple, les noms et prénoms), des dictionnaires ont été définis à l'aide des mêmes données que celles qui ont été utilisées pour générer les données d'entraînement de nos modèles d'apprentissage automatique. La recherche à l'aide des dictionnaires est d'abord effectuée de manière exacte et en cas d'échec de manière approximative en utilisant la distance Jaro-Winkler [Jar89, Win99]. Pour effectuer une prédiction, l'algorithme associe une classe à chaque élément de la colonne. Cette classe est déterminée en cherchant d'abord parmi les expressions régulières, puis en cas d'échec dans les dictionnaires. La classe finale prédite pour la colonne est celle qui est majoritaire parmi les éléments de la colonne.
Sur ce nouveau jeu de données, le taux de bonne reconnaissance pour la RF est de 0,97 et de 0,98 pour Catboost contre 0,91 pour le modèle utilisant des expressions régulières et des dictionnaires. La méthode utilisant l'apprentissage automatique montre un score parfait pour les types sémantiques

uniquement composés de caractères alphabétiques mais est en difficulté avec les types numériques. En particulier avec les numéros de téléphone qui sont confondus avec les devises. Cela est dû au fait que les caractéristiques utilisées ne sont pas parfaitement adaptées aux données de type numérique : [KG21].

Les résultats d'un test de la méthode sur les données de l'entreprise partenaire de la thèse est présentée dans l'annexe A.

3.3.7 Conclusion

Dans cette section, nous avons exposés les principaux freins à l'usage de méthodes d'apprentissage automatique au domaine de la détection de types sémantiques de manière industrielle. Afin de lever ces freins nous avons proposé une méthode utilisant un faible volume de données réelles et permettant l'utilisation de types sémantiques personnalisés (n'étant pas dans les jeux de données les plus courants).
De plus, nous avons proposé une simplification des caractéristiques utilisées dans l'état de l'art afin de les rendre plus polyvalentes sans compromettre l'efficacité de la méthode. Comme le problème de la détection automatique de types sémantiques peut être un problème de type "ensemble ouvert", nous avons introduit une solution pour gérer ce scénario. Des études complémentaires sont nécessaires sur plusieurs points. Tout d'abord, pour améliorer le taux de bonne reconnaissance, en particulier dans le cas où les éléments de la colonnes sont distribués de façon très déséquilibrée ou présentent un faible nombre de valeurs distinctes. Une première approche pourrait être de générer un plus grand nombre d'exemples avec des valeurs omega faibles. Cependant cela ne sera probablement pas suffisant pour gérer les cas où l'on peut retrouver une très grande répétition d'une même valeur, par exemple, le même nom dans toute une colonne. Dans ce cas, il serait intéressant d'étudier la possibilité d'utiliser une méthode hybride combinant apprentissage automatique, expressions régulières et dictionnaires. En privilégiant l'usage des expressions régulières et des dictionnaires quand le nombre de valeurs distinctes dans la colonne est faible. Il serait ensuite intéressant d'explorer des architectures dédiées à la reconnaissance d'ensembles ouverts afin d'améliorer les performances dans ce contexte [DGB18].

3.4 Détection des Colonnes quasi-dupliquées

3.4.1 Introduction

La recherche de documents quasi-dupliqués ou dupliqués est un thème de recherche classique et ancien [Bro00]. Dans le cas particulier des bases de données la "résolution d'entité" caractérise le fait de chercher à identifier des enregistrements différents caractérisant la même entité [PIP20] (on parle aussi de similarité entre les enregistrements : doubles exacts et similaires).

Cela peut s'effectuer à l'échelle de plusieurs bases de données mais s'effectue généralement au sein d'une même table afin d'entamer un processus de déduplication [IC19]. Ici nous n'allons pas nous intéresser aux lignes mais aux colonnes. L'objectif est de déterminer si deux colonnes sont quasi-dupliquées afin de pouvoir les étiquetter dans un outil comme DataHub [2].

S'intéresser à la similarité entre deux colonnes n'est pas nouveau mais était précédemment vu comme un outil [LHK04] pour identifier des enregistrements quasi-dupliqués.
L'originalité de la méthode présentée est de ne pas s'appuyer directement sur les colonnes, une métrique de similarité et un seuil fixé par un expert pour décider, mais d'utiliser un vecteur de metadonnées représentant la colonne, une méthode de génération de données et un algorithme d'apprentissage automatique pour identifier les colonnes quasi-dupliquées.

3.4.2 Description générale

Une colonne C isolée peut être décrite comme un multi-ensemble [AMS99], $C = (c_1, c_2, .., c_L)$ contenant L éléments. Chaque c_i est membre d'un univers contenant D valeurs possibles et où plusieurs éléments peuvent être identiques.

On définit une colonne quasi-dupliquée (Near Duplicate Column NDC) de C comme une version modifiée C dans laquelle un nombre inconnu β de c_i a été supprimé et un nombre inconnu γ d'éléments a été inséré dans C depuis D.

On note $\alpha = \frac{\beta+\gamma}{L}$

Une méthode traditionnelle pour évaluer la similarité entre deux ensembles est d'utiliser une fonction de similarité [Har20] comme la similarité de Jaccard [Jac12]. Cette fonction de similarité peut être adaptée de la manière suivante aux multi-ensembles [LRU20] :

$$J(A, B) = \frac{|A \cap B|}{|A| + |B|} \tag{3.4}$$

Où $\cap$ est l'intersection entre multi-ensembles. La fonction prend des valeurs entre 0 et 1/2. C'est un outil efficace pour mesurer la similarité mais souffre de quelques faiblesses. Le coût en calcul devient élevé pour des colonnes de grande dimension. Notre méthode devient plus rapide pour des colonnes d'environ 20000 éléments. De plus, cette fonction ne peut pas capturer toutes les situations possibles. Par exemple : si l'on définit $A = \{a, b, c\}$, $B = \{a, b, c, a, b, c, a, b, c\}$, $C = \{a, b, c, e, f, g, h\}$ on a $J(A, B) = J(A, C)$.

Ici nous allons utiliser une autre méthode pour comparer deux colonnes.

2. https ://datahubproject.io/

Cette méthode s'appuie sur l'extraction à partir de C de vecteurs de caractéristiques de longueur fixe à valeur dans $\mathbb{R}^n$ [HHB+19].

Ces vecteurs vont ensuite être utilisés pour entraîner un algorithme d'apprentissage automatique à distinguer entre colonnes quasi-dupliquées (NDC) ou non quasi-dupliquées (NNDC).

3.4.3 Description des caractéristiques

Les caractéristiques utilisées correspondent globalement à celles décrites dans la section 3.2. Cependant le nombre de caractéristiques est plus important car les majuscules sont prises en compte. De plus s'ajoute aux statistiques calculées le coefficient d'asymétrie ainsi que le kurtosis. Ce qui donne un total de 870 caractéristiques.

3.4.4 Principe des algorithmes

Pour être capable d'apprendre à distinguer entre les NDC et les NNDC, nous allons générer une population de NDC pour une colonne C spécifique ainsi qu'une population de NNDC similaire à cette dernière. Nous allons ensuite entraîner un classifieur à distinguer entre les deux classes.

Le processus de génération des NDC pour une colonne C spécifique est décrit dans l'algorithme 4. Dans cet algorithme, nous créons simplement une colonnes NDC pour une colonne spécifique C en appliquant α*length(C) modifications à C. Ces modifications sont des insertions (c'est-à-dire prendre une valeur dans D et l'insérer dans C) et des suppressions (supprimer aléatoirement une valeur de C). L'ordre dans lequel interviennent ces modifications est aléatoire.

La fonction GenerateNNDC n'est pas décrite car elle implique simplement de sélectionner aléatoirement l (égale au nombre d'éléments dans C) valeurs dans D.

Algorithm 4 GenerateNDC, Génère une colonne presque dupliquée

INPUT D, C, α
Initialisation $NDC \leftarrow copy(C)$
for $i \leftarrow 0$ **to** $round(\alpha * length(C))$ **do**
  $\kappa \leftarrow random(0, 1)$
  if $\kappa = 1$ **then**
    $\theta \leftarrow random(0, length(D) - 1)$
    $NNDC \leftarrow concat(NDC, D[\theta])$
  else
    if $length(NDC) > 0$ **then**
      $deleteNDC[random(0, length(NDC))]$
    else
  end
end
OUTPUT NDC

L'algorithme 5 décrit une méthode pour générer un jeu de données de NDC et de NNDC pour une colonne C spécifique avec N exemples de chaque classe. Pour ce faire, nous générerons N à l'aide de la colonne C et ensuite nous extrairons et stockerons les caractéristiques de ces NDC avec le label associé. Nous ferons simultanément la même chose pour les NNDC.

Algorithm 5 Génère un jeu de données de NDC et NNDC pour une colonne C spécifique

INPUT *D, C, N,* α
Initialisation : $xNNDC \leftarrow []$
$xNDC \leftarrow []$
$yNNDC \leftarrow []$
$xNDC \leftarrow []$
$l \leftarrow length(C)$
for $i \leftarrow 0$ **to** $N - 1$ **do**
    $NDC \leftarrow GenerateNDC(D, C, \alpha)$
    $NNDC \leftarrow GenerateNNDC(D, l)$
    $xNNDC[i] \leftarrow extractFeatures(NDC)$,
    $yNNDC[i] \leftarrow 0$
    $xNDC[i] \leftarrow extractFeatures(NDC)$
    $yNDC[i] \leftarrow 1$
end
$x \leftarrow concat(xNNDC, xNDC)$
$y \leftarrow concat(yNNDC, yNDC)$
OUTPUT x,y

L'algorithme 6 présente une méthode pour générer un jeu de données qui n'est pas construit pour une colonne C spécifique. Nous répétons N fois la même procédure. Tout d'abord, un D est tiré aléatoirement dans U (un ensemble de multi-ensembles). Ensuite, une colonne C est générée en sélectionnant des éléments dans D. L'étape suivante consiste à générer une NDC et une NNDC pour cette colonne C et à calculer le vecteur de caractéristiques de ces 3 colonnes.
Puis, nous utilisons ces caractéristiques pour en construire de nouvelles en suivant le procédé décrit dans la section généralisation. Un nouveau vecteur est ainsi construit pour le couple (C,NNDC) et un autre pour le couple (C,NDC). Ces caractéristiques finales sont ensuite stockées ainsi que leur label associé. On continue de répéter ces opérations jusqu'à obtenir le nombre d'exemples voulus.

Algorithm 6 Création d'un jeu de données de NDC et NNDC en général

INPUT U, N, α,
initialisation $xNNDC \leftarrow []$
$xNDC \leftarrow []$
$yNNDC \leftarrow []$
$yNDC \leftarrow []$
for $i \leftarrow 1$ **to** N **do**
  $\kappa \leftarrow ramdom(0, length(U))$
  $D \leftarrow U[\kappa]$
  $C \leftarrow GenerateNNDC(D, l)$
  $NDC \leftarrow GenerateNDC(D, C, \alpha)$
  $NNDC \leftarrow GenerateNNDC(D, l)$
  $xC \leftarrow extractFeatures(NDC)$
  $fNNDC \leftarrow extractFeatures(NDC)$
  $yNNDC[i] \leftarrow 0$
  $fNDC \leftarrow extract\_features(NDC)$
  $yNDC[i] \leftarrow 1$
  for $j \leftarrow 0$ **to** *length(xC)-1* **do**
    if $|fNNDC[j]| + |xC[j]| \neq 0$ **then**
      $tNNDC[j] \leftarrow \frac{|fNNDC[j]-xC[j]|}{|fNNDC[j]|+|xC[j]|}$
    else
      $tNNDC[j] \leftarrow 0$
    end
    if $|fNDC[j]| + |xC[j]| \neq 0$ **then**
      $tNDC[j] \leftarrow \frac{|fNDC[j]-xC[j]|}{|fNDC[j]|+|xC[j]|}$
    else
      $tNDC[j] \leftarrow 0$
    end
    $xNNDC[i] \leftarrow tNNDC \;\; xNDC[i] \leftarrow tNDC$
  end
end
$x \leftarrow concat(xNNDC, xNDC)$
$y \leftarrow concat(yNNDC, yNDC)$
OUTPUT x,y

3.4.5 Généralisation

La première version de l'algorithme que nous avons présentée se concentre sur comment apprendre à distinguer entre une NDC et une NNDC pour une colonne C spécifique. Cette méthode peut être généralisée pour n'importe quelle colonne C. C'est ce qui est décrit dans l'algorithme 6.
Tout d'abord, nous n'utilisons plus un seul multi-ensemble D, mais un ensemble de multi-ensembles noté U. Ici U est une liste de 15 D différents que nous avons collectés. Ces colonnes appartiennent aux types sémantiques suivant : adresses emails, pays, régions, noms, URL, Code postaux, animaux,

villes, noms de banques, noms de forêts, noms de films, noms de rivières, équipes de footballs, maladies, mangas.

Nous sélectionnons tout d'abord aléatoirement un D dans U et construisons une colonne C en prenant des valeurs aléatoires (entre 200 et 400) dans D. Pour cette colonne C, nous construisons une NDC et une NNDC et calculons les vecteurs de caractéristiques des colonnes *C,NNDC et NDC*.

Si l'on note V l'un de ces vecteurs, pour chaque caractéristique k on calcule :

$$V[k] = \frac{|NDC[k] - C[k]|}{|NDC[k]| + |C[k]|} \tag{3.5}$$

Nous construisons ainsi les vecteurs de caractéristiques pour les couples *(C,NDC)* et *(C,NNDC)*.

Cette formule correspond à appliquer la distance de camberra [LW66] sur chaque dimension des vecteurs.

Nous construisons ainsi pour chaque colonne C un exemple positif et un exemple négatif. Et nous utilisons ensuite ces exemples pour entraîner un classifieur à faire la distinction entre les deux.

3.4.6 Expériences

Toutes nos expériences sont lancées sur des notebook Colab avec un CPU Xeon 2.30GHz avec 4 cores et 25Go de Ram.

Notre première expérience utilise comme D une liste de 13000 adresses emails, C est construit en tirant de manière uniforme entre 400 et 600 valeurs dans D. La valeur de N est de 100 et celle α de 1. On reproduit ensuite la même expérience mais avec une liste de 16000 noms comme D et α=2.

Les résultats sont, ensuite, projetés dans un espace bi-dimensionnel avec la méthode t-SNE [vdMH08] afin de pouvoir visualiser la séparabilité entre les NDC et NNDC.

Dans les deux cas (Fig 3.10, Fig 3.11), cette représentation des NDC et des NNDC forme deux groupes de points facilement séparables. Comme prévu, la séparabilité est moins nette quand $\alpha = 2$ car les données ont davantage été modifiées. On peut donc espérer de bons résultats lors de l'utilisation des algorithmes d'apprentissage automatique pour séparer les deux.

Dans l'expérience suivante nous n'allons pas projeter les données dans un espace bi-dimensionnel, mais nous allons les utiliser pour entraîner une RF (avec les paramètres suivants : profondeur max 18, 200 estimateurs) à discerner entre les NDC et NNDC pour différents α.

Dans cette expérience nous allons utiliser comme D deux multi-ensembles : l'un avec des noms (16000) et l'autre avec des adresses emails (13000).

Les colonnes C sont générées en sélectionnant aléatoirement entre 300 et

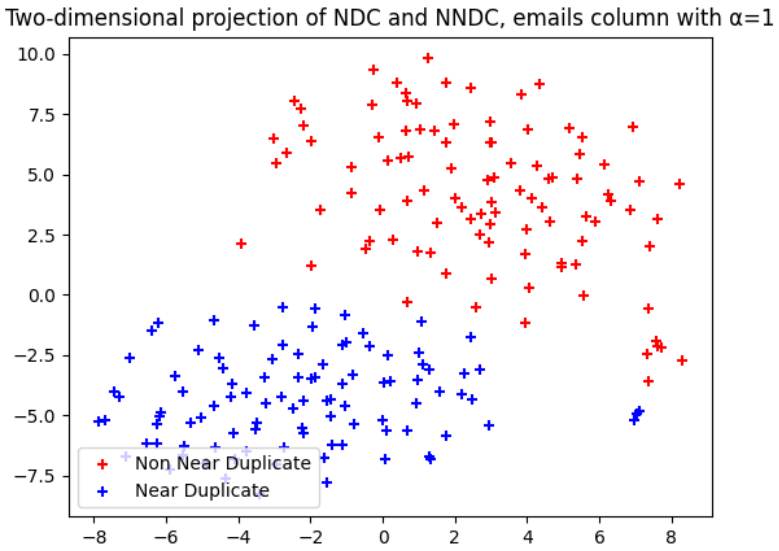


FIGURE 3.10 – Projection en deux dimensions de NDC et NNDC, pour des colonnes d'adresses emails avec $\alpha = 1$

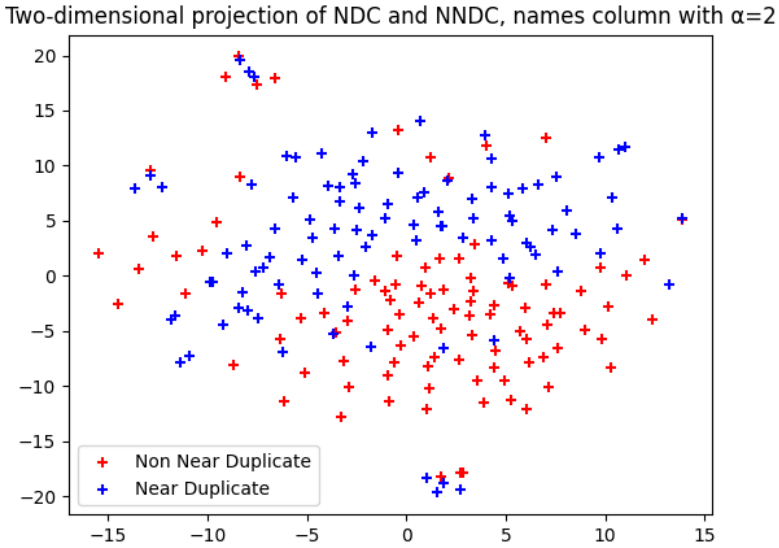


FIGURE 3.11 – Projection en deux dimensions de NDC et NNDC, pour des colonnes de noms avec $\alpha = 2$

600 valeurs dans D. Pour chaque niveau de alpha l'expérience est répétée 30 fois. N est choisi à 1000 et la moyenne du taux de bonne reconnaissance est calculée à l'aide d'une validation croisée 10 plis.

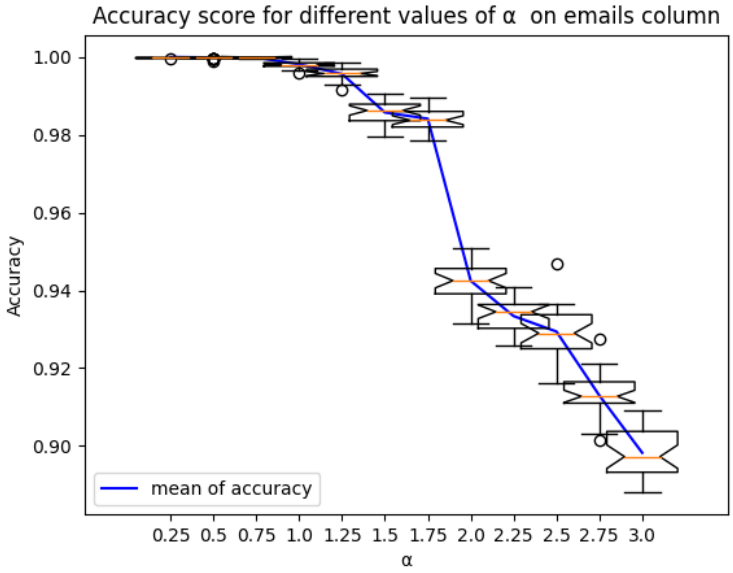


FIGURE 3.12 – Taux de bonne reconnaissance pour différentes valeurs de α sur des colonnes d'adresses emails

TABLE 3.9 – Taux de bonne reconnaissance pour la première expérience avec les colonnes de mails et de noms part 1.

α	0.25	0.5	0.75	1	1.25	1.5
mean name	0.99	0.99	0.99	0.98	0.97	0.94
var name	1e-07	3e-07	4e-07	4e-06	1e-05	2e-05
mean emails	0.99	0.99	0.99	0.99	0.99	0.98
var emails	8e-09	6e-08	4e-08	7e-07	2e-06	8e-06

Notre deuxième expérience utilise l'algorithme 3.4.4. Nous l'appliquons sur U afin de créer notre jeu de données d'apprentissage. Puis, nous utilisons une RF (200 estimateurs, profondeur maximum 18) et une Light Gradient Boosting Machine (LGMB) [KMF+17] (200 estimateurs, profondeur maximum 14, nombre de feuilles 12000) pour apprendre des données pour différentes valeurs de α (entre 0.25 et 1.25). Le taux de bonne reconnaissance est évalué en prenant le résultat moyen à l'aide d'une validation croisée 5 plis. La valeur de N est fixée à 2500.

Nous construisons ensuite (avec le même α) un nouvel ensemble U' (N est fixé à 500) avec 9 nouveaux D qui n'étaient pas présents dans U (types sémantiques : civilité, continent, marque de voiture, mer, monnaies, océan,

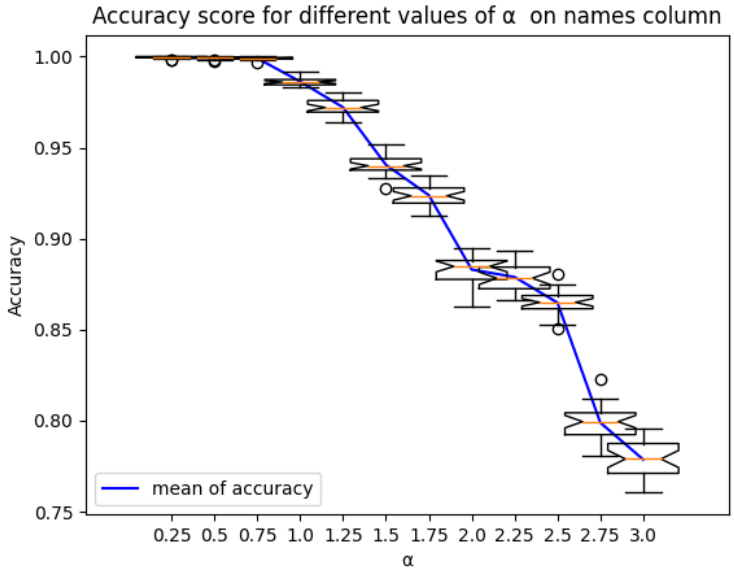


FIGURE 3.13 – Taux de bonne reconnaissance pour différentes valeurs de α sur des colonnes de noms

TABLE 3.10 – Taux de bonne reconnaissance pour la première expérience avec les colonnes de mails et de noms part 2.

α	1.75	2	2.25	2.5	2.75	3
mean name	0.92	0.88	0.87	0.86	0.79	0.77
var name	3e-05	5e-05	5e-05	4e-05	9e-05	9e-05
mean emails	0.98	0.94	0.93	0.92	0.91	0.89
var emails	7e-06	2e-05	1.e-05	3e-05	3e-05	3e-05

os, stades, université). Puis, nous entraînons 3 classifieurs différents sur U (N est fixé à 2500) : une RF (200 estimateurs, profondeur maximum 18), une LGBM (200 estimateur, maximum depth 14, nombre de feuilles 12000) et un perceptron multicouches (5 couches cachées avec relu, 0.7 de dropout, 0.0001 l2 régularisation). Enfin, nous testons ces modèles sur *U'*.

TABLE 3.11 – Taux de bonne reconaissance pour la deuxième expérience sur U pour 2 classifieurs

α	RF sur U	LigthGBM sur U
0.25	0.815	**0.949**
0.5	**0.982**	0.979
0.75	**0.963**	0.963
1	**0.936**	0.931
1.25	**0.914**	0.905

TABLE 3.12 – Taux de bonne reconnaissance sur U' pour 3 classifieurs

Classifieur / α	RF sur U'	LGBM sur U'	NN sur U'
0.25	**0.995**	0.951	0.769
0.5	0.792	**0.938**	0.727
0.75	0.775	**0.895**	0.686
1	0.734	**0.849**	0.651
1.25	0.704	**0.822**	0.614

TABLE 3.13 – Précision sur U' pour 3 classifieurs

Classifieur / α	RF sur U'	LGBM sur U'	NN sur U'
0.25	**0.995**	0.913	0.685
0.5	0.710	**0.910**	0.651
0.75	0.701	**0.844**	0.618
1	0.662	**0.790**	0.592
1.25	0.640	**0.768**	0.566

Dans notre dernière expérience nous utilisons à nouveau l'algorithme 3.4.4 pour entraîner une RF et une LGBM (avec les mêmes paramètres que précédemment) sur U. α est choisi égal à 0.25 puis 1.25 et enfin choisi aléatoirement entre (0 et 1.25) à chaque appel de la fonction *GenerateNDC*.

Nous évaluons les classifieurs sur des jeux de données de test créés à partir de *U'* avec des valeurs de alpha allant 0.25 à 1.25.

TABLE 3.14 – Rappel sur U' pour 3 classifieurs

Classifieur / α	RF sur U'	LGBM sur U'	NN sur U'
0.25	0.994	**0.996**	**0.996**
0.5	**0.986**	0.972	0.978
0.75	0.956	**0.968**	**0.968**
1	0.956	0.950	**0.968**
1.25	0.932	0.922	**0.966**

TABLE 3.15 – F1-score sur U' pour 3 classifieurs

Classifieur / α	RF sur U'	LGBM sur U'	NN sur U'
0.25	**0.994**	0.953	0.811
0.5	0.825	**0.940**	0.781
0.75	0.809	**0.902**	0.755
1	0.782	**0.862**	0.735
1.25	0.758	**0.838**	0.714

TABLE 3.16 – Taux de bonne reconnaissance pour 2 modèles entraînés sur plusieurs alpha pour la dernière expérience

α	0.25	0.5	0.75	1	1.25
0.25 RF	0.887	0.838	0.733	0.673	0.620
0.25 LGMB	**0.961**	0.9115	0.8155	0.7445	0.704
1.25 RF	0.822	0.81	0.800	0.785	0.778
1.25 LGMB	0.877	0.867	0.855	0.855	**0.839**
random RF	0.863	0.86	0.821	0.788	0.734
random LGMB	0.918	**0.919**	**0.894**	**0.862**	0.816

TABLE 3.17 – Rappel pour 2 modèles entraînés sur plusieurs alpha pour la dernière expérience

α	0.25	0.5	0.75	1	1.25
0.25 RF	0.995	0.894	0.684	0.545	0.457
0.25 LGMB	0.987	0.884	0.693	0.544	0.457
1.25 RF	0.998	**0.989**	**0.959**	0.94	**0.913**
1.25 LGMB	0.996	0.988	0.96	**0.943**	0.911
random RF	**1.0**	0.988	0.934	0.864	0.735
random LGMB	0.997	0.983	0.95	0.888	0.802

TABLE 3.18 – Précision pour 2 modèles entraînés sur plusieurs alpha pour la dernière expérience

α	0.25	0.5	0.75	1	1.25
0.25 RF	0.818	0.804	0.759	0.732	0.679
0.25 LGMB	**0.938**	**0.935**	**0.917**	**0.908**	**0.890**
1.25 RF	0.738	0.728	0.728	0.718	0.719
1.25 LGMB	0.804	0.795	0.794	0.801	0.797
random RF	0.784	0.786	0.762	0.75	0.7335
random LGMB	0.860	0.871	0.855	0.844	0.825

TABLE 3.19 – F1-score pour 2 modèles entraînés sur plusieurs alpha pour la dernière expérience

α	0.25	0.5	0.75	1	1.25
0.25 RF	0.898	0.846	0.719	0.625	0.546
0.25 LGMB	**0.961**	0.908	0.789	0.680	0.611
1.25 RF	0.849	0.838	0.827	0.814	0.804
1.25 LGMB	0.890	0.881	0.869	**0.866**	**0.850**
random RF	0.879	0.875	0.839	0.802	0.734
random LGMB	0.924	**0.923**	**0.900**	0.865	0.813

3.4.7 Résultats

Les premières expériences (Fig 3.12 et 3.13) confirment ce que l'on a observé sur les projections bi-dimensionnelles. Les NDC et NNDC sont facilement séparables si α est inférieur à 1 (Fig 3.12, 3.13 et tableaux 3.9, 3.10). En effet, dans cette plage de valeurs, le taux de bonne reconnaissance est proche de 99%. les résultats déclinent ensuite rapidement à cause du nombre très important de modifications qu'ont subi les données.
Ces résultats sont moins bons que ceux que l'on pourrait obtenir dans ce cas avec une méthode traditionnelle. En effectuant la même expérience mais en utilisant la similarité de Jaccard pour séparer les deux populations, nous pouvons obtenir des résultats parfaits jusqu'à α égal 3. Néanmoins, ces résultats prouvent que les caractéristiques que nous utilisons sont appropriées pour ce problème d'apprentissage. On peut aussi constater certaines limitations de notre méthodes : les résultats déclinent plus vite avec les colonnes de noms que les colonnes d'adresses emails. En effet, les colonnes de noms contiennent moins de caractères différents que les colonnes d'emails et les chaînes de caractères sont plus courtes, elles sont donc plus difficiles à caractériser avec nos vecteurs caractéristiques. Nous avons confirmé ces résultats en testant sur des chaînes très courtes avec peu de caractères différents (des groupes sanguins) et les résultats déclinent encore plus rapidement.

Pendant la deuxième expérience, nous avons utilisé la version généralisée de notre méthode décrite dans l'algorithme 3.4.4. Comme on peut s'y attendre les résultats sont moins bons que pour la version spécialisée de l'algorithme. Les meilleurs taux de bonne reconnaissance sur U sont obtenus en utilisant une RF et une LGBM. Les résultats moyens sur U avec validation croisée 5 plis sont tous supérieurs à 90%. On peut ainsi confirmer que notre méthode de calcul de caractéristiques est valide pour ce problème. De plus, ces résultats sont importants car dans cette configuration il est difficile de trouver un seuil pour une méthode basée sur la similarité de Jaccard.

Dans l'expérience suivante (Tables 3.4.6, 3.20, 3.13 et 3.15), nous avons entraîné nos classifieurs sur U et les avons testés sur U'. En effet, les données présentes dans U' sont très différentes de celles dans U. Nous avons effectué cette expérience car dans la réalité, les données que nous allons rencontrer seront très différentes de celles sur lesquelles l'apprentissage a été fait.

Comme on pouvait s'y attendre, les résultats sur U' sont inférieurs à ceux sur U, mais la LGBM présente des résultats très satisfaisants (au dessus de 90% de bonne reconnaissance) jusqu'à $\alpha = 0.75$. Ce déclin est logique car la capacité de l'algorithme à distinguer si deux colonnes sont NDC ou NNDC est liée au type sémantique des données qui ont servi pour l'apprentissage dans U. Cette capacité ne se transfère pas parfaitement sur U' où les catégories sont différentes de celles de U. Cependant notre modèle reste très efficace si α est faible. Nous avons aussi remarqué lors de nos tests que Ada-

boost [FS99] et Catboost [PGV+18] présentent de biens meilleurs résultats que la RF et le réseau de neurones. Il semble donc que ces algorithmes soient plus adaptés à cette tâche.

Dans la dernière expérience (Tables 3.16, 3.18, 3.17, 3.19), nous avons essayé de déterminer la meilleure méthode pour générer les données d'entraînement de nos modèles. Ceci est dans le but d'obtenir les meilleurs résultats sur *U'* (pour toutes les valeurs de α). On peut ainsi constater qu'entraîner avec des données modifiées avec un faible α donne de bons résultats sur les données de test uniquement si l'α utilisé pour générer les données de test est lui aussi faible. Quand les données de tests ont été générées avec un α plus grand il y a une baisse drastique du F1-score du modèle due à une forte diminution du rappel, cela montre l'incapacité du modèle à réaliser des classifications correctes. Entraîner à l'aide de données générées à partir d'une valeur élevée de α engendre des résultats moyens et stables sur toutes données de tests (ce qui est logique vu la méthode de génération), ainsi cette alternative surpasse la précédente.

Enfin, dans le dernier scénario nous avons utilisé des valeurs aléatoires de α (changeant à chaque création d'exemple) afin de générer les exemples.

Les algorithmes entraînés sur ce type de données surpassent les versions spécialisées pour une seule valeur de α. On peut ainsi conclure que créer les données d'apprentissage à l'aide d'une large gamme de α semble être la meilleure stratégie pour des applications réelles où α est inconnu.

3.4.8 Conclusion

Dans cette section nous avons introduit le concept de colonnes quasi-dupliquées. Nous avons aussi présenté une méthode pour détecter des colonnes quasi-dupliquées pour une colonne C spécifique à l'aide d'apprentissage automatique. Nous avons ensuite généralisé la méthode présentée pour qu'elle soit applicable sur n'importe quelle colonne C. Nous avons testé notre méthode sur des données réalistes et avons exploré plusieurs possibilités afin de créer les données d'apprentissage.

Afin d'améliorer les résultats de la méthode nous allons poursuivre nos recherches dans plusieurs directions. Tout d'abord une piste de recherche est d'effectuer un pré-traitement sur les données afin d'essayer d'uniformiser les données avant d'appliquer notre méthode. En effet la méthode est robuste à de faibles transformations des données mais se trouve vulnérable face à des transformations plus importantes (par exemple février transformé en fev ou 02). De plus il serait aussi intéressant de créer des données d'apprentissage à l'aide d'un U contenant un plus grand nombre de colonnes différentes. Enfin, il serait utile d'évaluer comment ce principe pourrait être généralisé à des sources de données non structurées.

3.5 La détection des jeux de données quasi-dupliquées

3.5.1 Introduction

Cette section s'inscrit dans la continuité de la section précédente. En effet, nous allons ici étendre le principe que nous avons utilisé sur les colonnes aux jeux de données. On va ainsi chercher à détecter si deux jeux de données sont quasi-dupliqués en représentant chaque jeu de données à l'aide d'un vecteur de métadonnées.

3.5.2 Description du problème

Nous considérons une source de données DS comme une instance I d'un schéma relationnel inconnu R, composé de K colonnes.

Chaque enregistrement est appelé un tuple t (une ligne). Chaque colonne C de l'ensemble de données peut être considérée comme un multi-ensemble [AMS99].
$C = (e_1, e_2, .., e_L)$ contient L éléments. Nous appelons L la longueur de la colonne. Chaque e est un élément d'un multi-ensemble D appelé domaine. Un ensemble de données quasi-dupliquées (NDDS) est défini comme une version modifiée d'un ensemble de données existant. Dans cette nouvelle version, θ colonnes ont été ajoutées et ι supprimées.

Nous désignons $\zeta = 10 * \frac{\theta+\iota}{K}$
De plus, bêta lignes ont été supprimées et gamma lignes insérées.
Nous désignons par $\alpha = \frac{\beta+\gamma}{L}$.
Le problème consiste à identifier si une DS est une version quasi-dupliquée d'un autre ensemble de données dont les paramètres α et ζ sont inconnus.

3.5.3 Caractéristiques

Nous allons utiliser le même principe que dans les sections précédentes pour générer les caractéristiques. Cependant, comme nous traitons cette fois-ci des jeux de données contenant plusieurs colonnes nous allons concaténer toutes les colonnes en une seule, le processus est illustré dans la figure 3.14. Le nombre de caractéristiques total est de 498 car nous n'avons utilisé que les minuscules et un sous-ensemble de caractères spéciaux.

De plus dans l'algorithme 9, nous aurons besoin de fusionner deux vecteurs de caractéristiques en un seul. Nous allons ici utiliser deux méthodes, celle qui a été utilisée pour la détection des colonnes quasi-dupliquées et une autre (qui n'avait pas été utilisée dans le cas des colonnes car elle donnait de moins bonnes performances). Si on a deux vecteurs v_1, v_2 et un vecteur fusionné V avec k variant de 1 au nombre de dimension :

La première possibilité se note :

$$\text{if } |v_1[k]| + |v_2[k]| \neq 0 \; V[k] = \frac{|v_1[k] - v_2[k]|}{|v_1[k]| + |v_2[k]|} \text{ else } V[k] = 0 \tag{3.6}$$

La deuxième :

$$V[k] = |v_1[k] - v_2[k]| \tag{3.7}$$

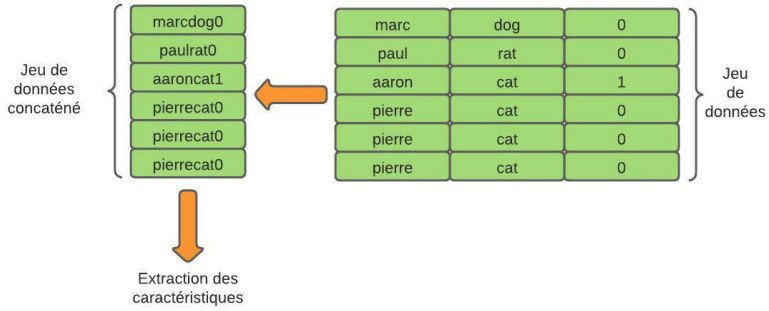


FIGURE 3.14 – Concaténation des colonnes avant extraction des caractéristiques

3.5.4 Algorithmes

L'algorithme 7 génère un ensemble de données à partir d'un ensemble de colonnes nommé univers. Le jeu de données tabulaire construit de cette manière compte nbr_colonnes colonnes et nbr_lignes lignes. La fonction pick_in_universe() choisit aléatoirement une colonne dans l'univers (et renvoie l'index de cette colonne dans l'univers). La fonction generate_col_from_ori choisit aléatoirement nbr_lignes dans une liste d'éléments. Nous utilisons cette fonction pour générer une colonne de la taille souhaitée à partir d'une grande liste d'éléments.
L'algorithme 8 crée une version quasi-dupliquée d'un ensemble de données existant. La fonction concat_elem concatène pour chaque ligne de l'ensemble de données tous les éléments de la ligne. Ainsi, lorsque nous appelons cette fonction, l'ensemble de données est réduit à une seule colonne. La fonction random_line ajoute et supprime des lignes à un jeu de données concaténé de cette façon. Nous allons utiliser cet algorithme afin de générer des exemples de NDDS pour nos ensembles d'apprentissage et de test.

L'algorithme 9 génère un exemple de chaque classe. La fonction make_features extrait le vecteur de caractéristiques temporaire de chaque ensemble de don-

Algorithm 7 RandomDataset génère un jeu de données à l'aide d'une liste de colonnes appelée univers. L'algorithme renvoie aussi les indices des colonnes utilisées.

input : *universe, nbr_col, nbr_line*
initialisation : $DataSet \leftarrow []$
$index\_universe \leftarrow []$
for $i \leftarrow 1$ **to** nbr_col **do**
  $original, index \leftarrow pick\_in\_universe(\ universe)$
  $index\_universe[i] \leftarrow index$
  $DataSet[i] \leftarrow generate\_col\_from\_ori(\ original, nbr\_line)$
end
OUTPUT DataSet,index_universe

Algorithm 8 AlterateDataset génère une version quasi-dupliquée d'un jeu de données en ajoutant et supprimant des colonnes et des lignes à celui-ci. La fonction renvoie la version quasi-dupliquée (concaténé) et le nombre de colonne utilisées pour construire ce jeu

input : *universe, Dataset ,list_index,* α *, nbr_modif*
initialisation : $cDataset \leftarrow copy(Dataset)$
$index\_universe \leftarrow []$
$clist\_index \leftarrow copy(list\_index)$
$nbr\_alteration \leftarrow round(\alpha * length(cDataset))$
for $i \leftarrow 0$ **to** nbr_modif **do**
  if $Random(True, False)$ **then**
    $original, index \leftarrow pick\_in\_universe(universe)$
    add $generate\_col\_from\_ori(original,\ length(cDataset[0]))$ to cDataset
    add index to $index\_universe$
  else
    $r = random(0, length(cDataset) - 1)$
    delete $cDataset[r]$
    delete $index\_universe[r]$
  end
end
$cDataset = concat\_elem(cDataset)$
for $i \leftarrow 0$ **to** $nbr\_alteration$ **do**
  if $Random(True, False)$ **then**
  else
    add $random\_line(universe, index\_universe)$ to cDataset
  end
  randomly delete an element from cDataset
end
OUTPUT cDataset, length($index\_universe$)

Algorithm 9 GenerateExemple returns two features vectors, one for a couple (DS,NNDDS) and the other for the couple (DS,NDDS)

INPUT : $universe,\ nbr\_col,\ \alpha,\ \zeta,\ minline,\ maxline$

$Dataset, indice\_universe \leftarrow RandomDataset(universe, nbr\_col, RandBetween(minline, maxline))$

$alterateDS, len\_indice \leftarrow AlterateDataset(universe, Dataset, indice\_universe, \alpha round((\zeta/10) * nbr\_col)$
$negDs, useless \leftarrow RandomDataset(\ universe, len\_indice, RandBetween(minline, maxline))$
$Dataset \leftarrow concat\_elem(Dataset)$
$negDs \leftarrow concat\_elem(negDs)$
$x\_pos \leftarrow make\_features(Dataset, alterateDS)$
$x\_neg \leftarrow make\_features(Dataset, negDs)$
OUTPUT x_pos, x_neg

nées et construit le vecteur de caractéristiques final en utilisant la formule 3.6 ou la formule 3.7.

Afin de valider la pertinence de notre méthode, nous utilisons l'algorithme 9 dans une boucle. Nous générons ainsi 1000 exemples de chaque classe (avec la formule 3.6) puis nous les projetons dans un espace bi-dimensionnel en utilisant t-SNE [vdMH08] (perplexité : 30). Le résultat est représenté dans la fig 3.15.

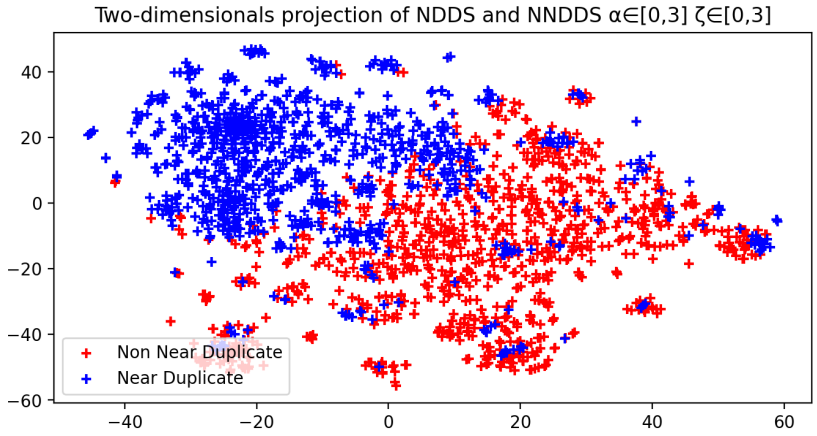


FIGURE 3.15 – NNDDS et NDDS pour $\alpha \in [0, 3]$ et $\zeta \in [0, 3]$, 10 colonnes avec la formule 3.6

Nous pouvons observer que les deux classes forment deux groupes distincts de points dans cet espace de faible dimension. Cela signifie que les caractéristiques que nous avons utilisées sont pertinentes et que nous pouvons les utiliser pour séparer les NNDDS des NDDS.

3.5.5 Expériences

Notre configuration expérimentale est un notebook Google Colab [Bis19] avec un processeur Xeon 2.30GHz 4 cœurs, 25Go de Ram et un tesla P100 (16go).

Dans toutes nos expériences, nous utiliserons l'algorithme 9 dans une boucle afin de générer nos données d'apprentissage et de test. Pour l'ensemble d'apprentissage, l'univers utilisé est composé de 100 colonnes contenant chacune un minimum de 8000 éléments. Pour l'ensemble de test, l'univers utilisé est constitué de 50 colonnes contenant chacune un minimum de 8000 éléments. Ces deux univers proviennent d'une collecte manuelle de données sur le jeu de données kaggle [3].

Dans la plupart des expériences, s'il n'est pas spécifié, le nombre de lignes pour chaque ensemble de données est choisi aléatoirement entre 50 et 300.

3.5.5.1 Influence du paramètre ζ

Notre première expérience utilise l'algorithme 9 afin de générer 7500 exemples de chaque classe avec les paramètres suivants : α égal à 2, nombre de colonnes (nbr_col) égal à 10, ζ allant de 0 à 3 ou choisi aléatoirement entre 0 et 4 (à chaque création de couple d'exemples). Dans chaque scénario, une RF est entraînée (200 estimateurs, profondeur maximale 18) pour distinguer les NNDS des NDDS. Les tests sont effectués sur des ensembles de données générés de la même manière (sans la version avec la sélection aléatoire de ζ et à partir de colonnes prévues pour former ces jeux de données de test) mais avec seulement 500 exemples de chaque classe. L'objectif est de déterminer la meilleure valeur du paramètre ζ à utiliser.

3.5.5.2 Influence du nombre de colonnes

Dans notre seconde expérience, nous allons explorer l'influence du nombre de colonnes utilisées pour construire les exemples. Nous allons générer plusieurs ensembles d'apprentissage et de test avec les paramètres suivants : α choisi aléatoirement entre 0 et 3 (pour chaque exemple généré), ζ choisi aléatoirement entre 0 et 4 (pour chaque exemple généré). Le nombre de colonnes varie : 5, 10, 20 ou choix aléatoire entre 5 et 22 (pour chaque exemple) pour l'ensemble d'apprentissage ; 5, 8, 11, 14, 17 et 20 pour l'ensemble de test. Le nombre d'exemples ainsi que le classifieur sont identiques à ceux de l'expérience 3.5.5.1.

3.5.5.3 Évaluation de plusieurs classifieurs

Dans notre troisième expérience, nous générons notre ensemble d'apprentissage en utilisant les paramètres suivants : α choisi aléatoirement entre 0

3. https ://www.kaggle.com/datasets

et 3 (pour chaque exemple généré), ζ choisi aléatoirement entre 0 et 4 (pour chaque exemple généré), le nombre de colonnes choisi aléatoirement entre 5 et 22 (pour chaque exemple généré). Pour les jeux de test, nous utilisons les paramètres suivants : α et ζ variant entre 0 et 3, le nombre de colonnes choisies aléatoirement entre 5 et 22 (pour chaque exemple généré). Les six classifieurs que nous utilisons sont Adaboost [FS99] (500 estimateurs), une Light Gradient Boosting Machine (LGBM) [KMF+17], (200 estimateurs, profondeur maximale 14, nbre de feuilles 12000), Catboost [PGV+18] (3000 itérations, profondeur 4, taux d'apprentissage 0, 1), une RF (200 estimateurs, profondeur max 18), TabNet [AP21](n_d 8, n_a 8, n_steps 3) et un stacking des 5 classifieurs précédents (cross_validation 3,Meta-Model : régression logistique) [Wol92].

3.5.5.4 Optimisations

Dans cette expérience, nous allons essayer d'améliorer le temps d'exécution de l'algorithme. Tout d'abord le processus d'extraction de caractéristiques nécessite de compter les caractères présents dans chaque élément, le temps de calcul induit par ce décompte augmente avec le nombre de lignes dans le jeu de données et le nombre de caractères à compter. Cette augmentation est observable dans la figure 3.4. Ainsi, lorsque le jeu de données a plus de 100000 lignes, le temps de calcul devient un problème (l'algorithme devient difficilement utilisable si l'on a de nombreux jeux de données à traiter)

Pour surmonter ce problème, nous proposons deux méthodes alternatives d'extraction de caractéristiques afin de réduire le temps de calcul. Dans la première, nous n'extrairons pas directement les caractéristiques sur l'ensemble des données concaténées mais sur des échantillons de celles-ci. Ensuite, pour chaque échantillon, nous extrairons les caractéristiques comme précédemment. Notre vecteur de caractéristiques final est alors construit en prenant la moyenne des résultats obtenus sur les échantillons dimension par dimension. Nous obtenons ainsi un vecteur de la même taille que précédemment, nous appellerons cette technique "méthode 1".

La deuxième méthode consiste simplement à calculer des caractéristiques sur un échantillon de l'ensemble des données, nous l'appellerons "méthode 2". Il est important de remarquer que certaines caractéristiques (comme celles utilisant les fonctions maximum et minimum) ne pourront pas être estimées de manière précise du fait de l'échantillonnage.

Les paramètres expérimentaux sont les mêmes que dans l'expérience 3.5.5.3. Cependant, nous modifions le nombre de lignes des jeux de données générées, qui est choisie aléatoirement entre 2000 et 3000 afin de simuler des jeux de données de plus grande taille. Pour la taille des échantillons nous utilisons un échantillon de taille 900 pour la "méthode 2" et 30 échantillons de tailles 30 pour "la méthode 1".

3.5.6 Résultats

Cette partie présente les résultats de toutes les expériences décrites dans la section expérience.

3.5.6.1 Influence du paramètre ζ

TABLE 3.20 – Taux de bonne reconnaissance pour plusieurs valeurs de ζ

Appr ζ \ Test ζ	0	1	2	3	moy
0	**1**	0.764	0.649	0.578	0.748
1	0.999	0.994	0.941	0.876	0.952
2	0.999	**0.998**	0.984	0.951	0.983
3	0.997	0.997	**0.989**	**0.971**	**0.988**
random(0,4)	**0.998**	0,998	0,985	0.954	0.984

Le tableau 3.20 contient les résultats de la première expérience. Tout d'abord, nous pouvons observer que le choix d'une faible valeur de ζ afin de générer l'ensemble d'apprentissage conduit à un faible taux de bonne reconnaissance. Ceci est normal car l'algorithme fait face à des situations totalement inédites (face aux individus générés avec un ζ élevé). Au contraire, lorsque ζ est élevé, les résultats sont bons dans toutes les situations, le système ayant déjà vu des exemples équivalents à une valeur inférieure de ζ (en raison du fonctionnement de l'algorithme 8). Dans la section 3.4 (qui ne porte que sur les colonnes), nous avons découvert que l'utilisation d'un α aléatoire (entre 0 et 3) pour générer les exemples d'apprentissage conduit à un meilleur taux de bonne reconnaissance [CBG+21]. Il est donc intéressant d'observer que ce n'est pas le cas pour le paramètre ζ, choisir une valeur de ζ égale 3 conduit à des résultats similaires voir meilleurs que l'utilisation d'un ζ choisi de manière aléatoire.

3.5.6.2 Influence du nombre de colonnes

Les résultats de la deuxième expérience dans le tableau 3.21 indiquent que le nombre de colonnes utilisé pour construire les exemples pour l'ensemble d'apprentissage n'a que peu d'influence sur les résultats (dans notre scénario). C'est intéressant car cela signifie que nous pouvons avoir de bons résultats sans avoir à créer des exemples pour chaque nombre de colonnes possible.

3.5.6.3 Évaluation de plusieurs classifieurs

Les tableaux 3.22 à 3.24 et les fig 3.16 à 3.27, présentent tous les résultats de la troisième expérience. Pour chaque expérience le même schéma se

TABLE 3.21 – Taux de bonne reconnaissance en fonction du nombre de colonnes utilisées pour construire le jeu de données d'entraînement et le jeu de données de test.

Appr nbr col \ Test nbr col	5	8	11	14	17	20	Moy
5	.967	.955	.968	.967	.971	**.978**	.968
10	.967	**.968**	**.971**	**.976**	.976	.976	**.972**
20	.961	.962	**.971**	.974	**.979**	**.978**	.971
random(5,22)	**.969**	.965	.965	.973	.974	**.978**	.970

dessine sur les cartes thermiques, une dégradation progressive des résultats avec l'augmentation des paramètres α et ζ. Les effets de l'augmentation de α sont assez faibles comparés à ceux de ζ. En effet, un ensemble de données contient généralement beaucoup plus de lignes que de colonnes, de sorte que les changements au niveau des colonnes ont un impact plus important sur les caractéristiques que nous extrayons que les changements au niveau des lignes. Cela est particulièrement vrai lorsque le nombre de colonnes est faible. Nous pouvons constater que l'utilisation de la formule 3.6 ou de la formule 3.7 a peu d'influence sur les résultats de la plupart des classifieurs. Seuls la RF et TabNet montrent une diminution notable du taux de bonne reconnaissance avec l'utilisation de la formule 2. Malgré l'utilisation d'une implémentation spécialisée pour les données tabulaires, le réseau neuronal montre des performances inférieures à celles des autres classifieurs. La RF présente des résultats légèrement inférieurs à ceux des algorithmes de gradient boosting. En effet, bien que ses résultats soient les meilleurs lorsque zeta est faible, ce classifieur souffre d'une plus grande perte de taux de bonne reconnaissance que les autres pour des valeurs de zeta plus élevées. Les algorithmes de gradient boosting ont des résultats similaires, mais Catboost est l'algorithme avec les meilleurs résultats globaux. Enfin, nous pouvons constater que le stacking dans ce problème n'apporte aucun avantage.

Les tableaux 3.26 et 3.27 présentent les caractéristiques les plus importantes lors de la prise de décision pour Catboost et la RF. Tout d'abord, nous pouvons constater que la caractéristique la plus importante dans tous les cas est l'entropie. En effet, elle va décrire globalement la répartition des données et donc tout changement de répartition des données dans la colonne va entraîner une modification de l'entropie.

Ensuite, nous pouvons remarquer la présence d'un grand nombre de statistiques sur les caractères de séparation tels que ".", '/' ou "-" ; ces caractères

TABLE 3.22 – Taux de bonne reconnaissance de six classifieurs pour différents couples de (α,ζ) avec la formule 3.6 part 1.

(α,ζ) / modeles	(0,0)	(1,0)	(2,0)	(3,0)	(0,1)	(0,2)	(0,3)	(1,1)	(1,2)
Catboost	.992	.995	.992	.993	.992	**.99**	.981	.996	.989
RF	**.994**	.996	**.994**	**.995**	**.995**	.981	.966	**.998**	.987
LGBM	.991	.995	.991	.992	.992	.988	.984	.995	.987
TabNet	.965	.968	.975	.969	.966	.948	.926	.957	.939
Adaboost	.99	.994	**.994**	.991	.989	.988	**.986**	.993	.983
Stacking	.992	**.997**	.993	.993	.993	.989	.982	.996	**.99**

TABLE 3.23 – Taux de bonne reconnaissance de six classifieurs pour différents couples de (α,ζ) avec la formule 3.6 part 2.

(α,ζ) / modeles	(1,3)	(2,1)	(2,2)	(2,3)	(3,1)	(3,2)	(3,3)	mean
Catboost	**.983**	.99	**.984**	**.98**	.993	.984	.971	.9874
RF	.967	**.992**	.979	.96	**.996**	.978	.953	.9830
LGBM	.982	.989	**.984**	.971	.99	.979	.968	.9859
TabNet	.924	.95	.930	.908	.96	.925	.91	.9446
Adaboost	.981	.991	.982	.973	.992	.98	**.977**	.9861
Stacking	.982	**.992**	**.984**	.974	.994	**.985**	.97	**.9876**

TABLE 3.24 – Taux de bonne reconnaissance de six classifieurs pour différents couples de (α,ζ) avec la formule 3.7 part 1.

(α,ζ) / modeles	(0,0)	(1,0)	(2,0)	(3,0)	(0,1)	(0,2)	(0,3)	(1,1)	(1,2)
Catboost	.993	.992	.99	.991	.994	.988	.984	.991	**.99**
RF	**.997**	**.999**	**.997**	**.997**	.992	.97	.94	.986	.966
LGBM	.996	.995	.994	.995	.996	**.99**	**.979**	.993	.989
TabNet	.995	.992	.99	.979	.973	.938	.897	.957	.923
Adaboost	.996	.998	.996	.995	.996	.988	.971	.993	.985
Stacking	**.997**	.998	**.997**	**.997**	**.998**	.986	.968	**.994**	.985

TABLE 3.25 – Taux de bonne reconnaissance de six classifieurs pour différents couples de (α,ζ) avec la formule 3.7 part 2.

modeles \ (α,ζ)	(1,3)	(2,1)	(2,2)	(2,3)	(3,1)	(3,2)	(3,3)	Moy
Catboost	**.982**	**.989**	**.986**	**.975**	.99	**.983**	**.973**	**.986**
RF	.945	.983	.966	.93	.982	.962	.915	.970
LGBM	.979	.989	.981	.968	.99	.98	.96	.985
TabNet	.9	.945	.913	.876	.948	.903	.856	.936
Adaboost	.973	.986	.978	.961	.991	.978	.96	.983
Stacking	.964	**.993**	.98	.957	**.992**	.975	.946	.982

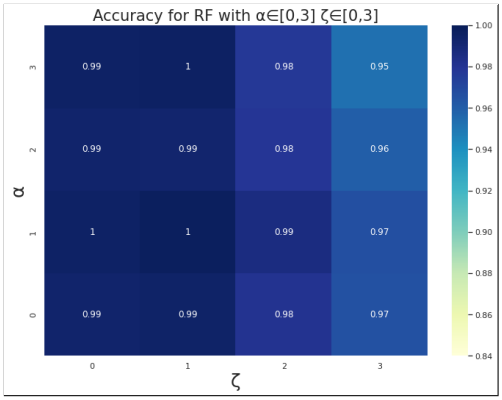


FIGURE 3.16 – Taux de bonne reconnaissance, RF classifieur pour $\alpha \in [0,3]$ et $\zeta \in [0,3]$ avec la formule 3.6

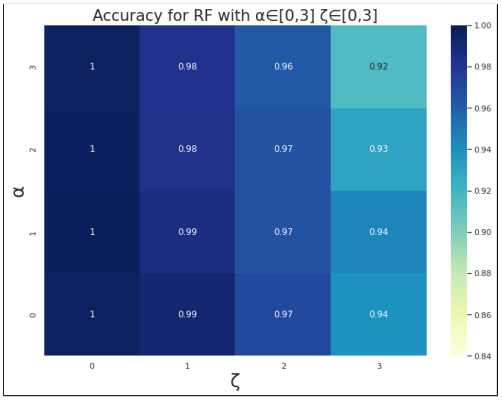


FIGURE 3.17 – Taux de bonne reconnaissance, RF classifieur pour $\alpha \in [0,3]$ et $\zeta \in [0,3]$ avec la formule 3.7

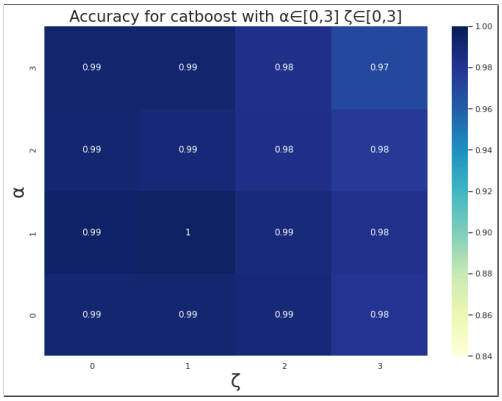


FIGURE 3.18 – Taux de bonne reconnaissance, Catboost classifieur pour $\alpha \in [0,3]$ et $\zeta \in [0,3]$ avec la formule 3.6

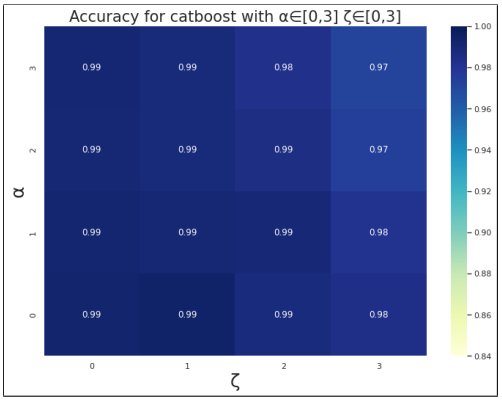


FIGURE 3.19 – Taux de bonne reconnaissance, Catboost classifieur pour $\alpha \in [0,3]$ et $\zeta \in [0,3]$ avec la formule 3.7

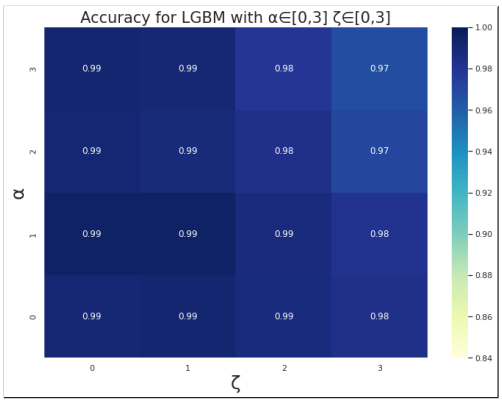


FIGURE 3.20 – Taux de bonne reconnaissance, LGBM classifieur pour $\alpha \in [0,3]$ and $\zeta \in [0,3]$ avec la formule 3.6

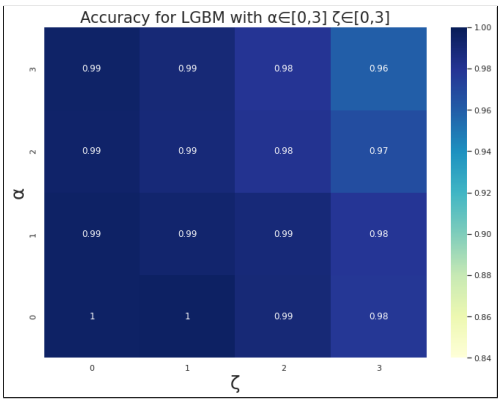


FIGURE 3.21 – Taux de bonne reconnaissance, LGBM classifieur pour $\alpha \in [0,3]$ and $\zeta \in [0,3]$ avec la formule 3.7

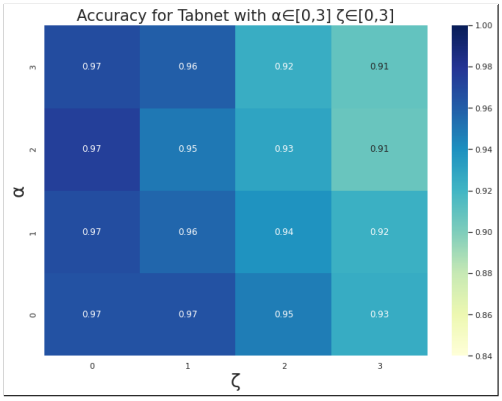


FIGURE 3.22 – Taux de bonne reconnaissance, TabNet classifieur pour $\alpha \in [0,3]$ and $\zeta \in [0,3]$ avec la formule 3.6

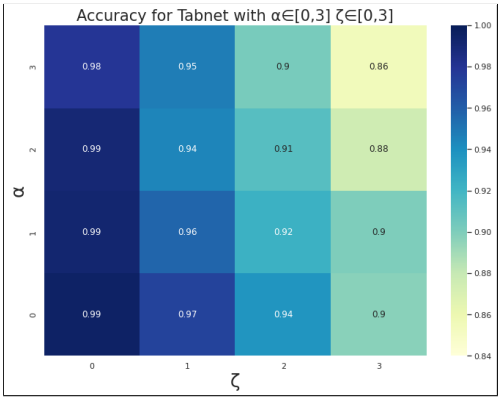


FIGURE 3.23 – Taux de bonne reconnaissance, TabNet classifieur pour $\alpha \in [0,3]$ et $\zeta \in [0,3]$ avec la formule 3.7

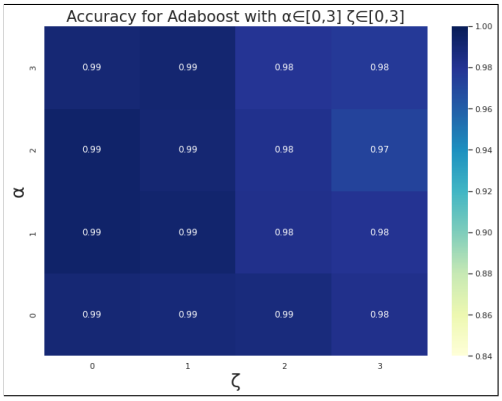


FIGURE 3.24 – Taux de bonne reconnaissance, Adaboost classifieur pour $\alpha \in [0,3]$ et $\zeta \in [0,3]$ avec la formule 3.6

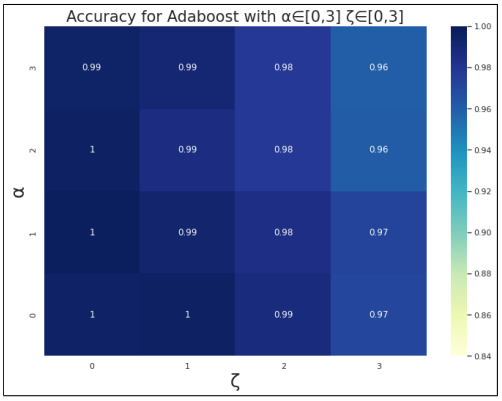


FIGURE 3.25 – Taux de bonne reconnaissance, Adaboost classifieur pour $\alpha \in [0,3]$ et $\zeta \in [0,3]$ avec la formule 3.7

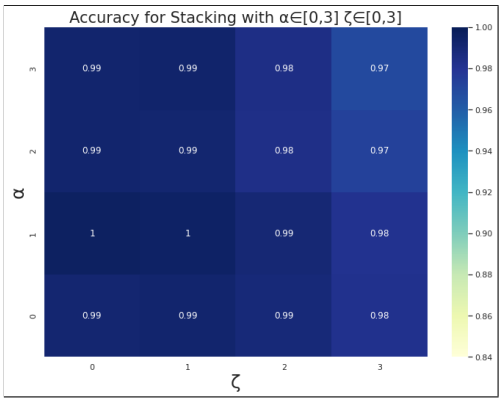


FIGURE 3.26 – Taux de bonne reconnaissance, Stack classifieur pour $\alpha \in [0,3]$ et $\zeta \in [0,3]$ avec la formule 3.6

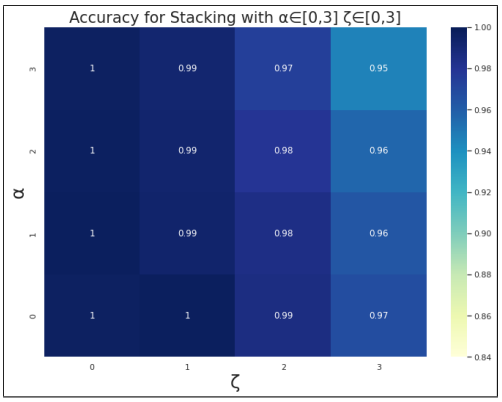


FIGURE 3.27 – Taux de bonne reconnaissance, Stack classifieur pour $\alpha \in [0,3]$ et $\zeta \in [0,3]$ avec la formule 3.7

sont très présents dans un petit nombre de types sémantiques et très peu dans les autres ce qui les rend très distinctifs. Enfin, on remarque dans ce top 10 la présence de nombreuses statistiques qui concernent des lettres peu fréquentes en anglais (la langue de nos exemples) par exemple 'z' [Ohl59], mais aussi du chiffre 9 qui est un chiffre rarement présent dans les données réelles [Ben38]. Cela rend un changement dans le nombre de ces caractères très distinctif.

TABLE 3.26 – Top 10 des caractéristiques pour les modèles Catboost et RF avec la formule 3.6, classé par rang descendant.

Catboost formule 3.6	RF formule 3.6
Entropie	Entropie
Moy nombre de 'f'	Moy nombre de ','
Nombre median de '.'	Moy nombre of 'z'
Moy nombre de '-'	Moy nombre de 'q'
Var nombre de '.'	Var nombre de ','
Nombre median de '0'	Var nombre de 'q'
Moy nombre de '/'	Moy nombre de '/'
Moy nombre de '+'	Moy nombre de '-'
Var nombre de '9'	Var nombre de '0'
Moy nombre of '-'	Moy nombre of 'f'

Après toutes ces expériences, nous pouvons conclure que la formule 3.7 est la meilleure car elle donne des résultats très similaires à la formule 3.6 tout en étant plus facile à calculer. De plus, dans d'autres expériences que nous avons réalisées avec plus de similarité entre les exemples positifs et négatifs. Par exemple en utilisant un alpha égal à 5,les modèles utilisant la formule 3.7 continuent à avoir les mêmes résultats alors que ceux utilisant la formule 3.6 voient leur résultat diminuer.

TABLE 3.27 – Top 10 des caractéristiques pour les modèles Catboost et RF avec la formule 3.7, classé par rang descendant.

Catboost formule 3.7	RF formule 3.7
Entropie	Entropie
Std of numerical chars cells	Std nbr num char dans cellules
Moy nombre de '-'	Moy nombre de ','
Moy nombre de '.'	Moy nombre de '-'
Moy nombre de 'f'	Var nombre de '.'
Moy nombre de 'q'	Var nombre de 'f'
Std nbr mot dans cellule	Moy nbr mot dans cellules
'.' dans toutes les cellules	Std nbr alphab char dans cellules
Moy nombre of '/'	Max nombre de ','
Longueur médiane chaînes de caractères	Max nombre de '0'

Par ailleurs, il faut noter que ces résultats dépendent de la méthode de génération et de test, que tous les cas ne peuvent être couverts et que les algorithmes doivent être adaptés à chaque situation. Par exemple, en modifiant la fonction RandomDataset (algorithme 7), on peut générer deux jeux de données avec le même nombre de lignes et dont les colonnes ont le même type sémantique. De plus, comme ces colonnes sont échantillonnées de la même manière à partir des colonnes présentes dans l'univers, elles sont extrêmement similaires sans pour autant être dupliqués. En utilisant cette méthode pour créer 1000 exemples négatifs à ajouter au jeu de données d'apprentissage, nous pouvons modifier les résultats sur les données de test. Dans ce cas, le taux de bonne reconnaissance de Catboost sur notre jeu de données de test précédent tombe à 93,8% alors qu'il est de 97,1% sur les exemples générés de la manière que nous venons de décrire (taux de bonne reconnaissance calculé sur les exemples générés à partir des données de test). Comme il n'existe pas de vérité terrain, il est nécessaire d'adapter la méthode de génération des exemples d'apprentissage en fonction des données réelles attendues.

3.5.6.4 Optimisations

Comme dans nos tests précédents, Catboost à les meilleurs résultats dans tous les scénarios. La méthode 2, bien que plus facilement parallélisable, donne des résultats inférieurs à la méthode 1. Il est particulièrement intéressant de noter que la perte de taux de bonne reconnaissance est faible avec Catboost. Ainsi, en utilisant la méthode 1 et Catboost on peut diviser par deux le temps de calcul tout en ne perdant que 0,3% de taux de bonne reconnaissance. Ces résultats doivent cependant être considérés avec prudence car ils sont fortement dépendants de la méthode de génération des données, notre algorithme de génération d'exemples ne génère pas tous les cas pos-

TABLE 3.28 – Taux de bonne reconnaissance pour 6 classifieurs en utilisant les deux méthodes et les deux formules

formula / models	3.6	3.7
Catboost m1	0.9835	**0.9852**
Catboost m2	0.9615	0.9711
RF m1	**0.9673**	0.9573
RF m2	0.9560	0.9528
LGBM m1	**0.9774**	**0.9774**
LGMB m2	0.9566	0.9660
TabNet m1	**0.8883**	0.8412
TabNet m2	0.8343	0.8700
Adaboost m1	0.9741	**0.9750**
Adaboost m2	0.9442	0.9638

sibles (la méthode de génération favorisant ensuite une prise de décisions de l'algorithme plus orienté sur les lignes ou les colonnes). Ainsi, en modifiant la méthode de génération comme décrit à la fin de la section précédente les résultats peuvent être sensiblement différents. La taille de l'échantillon étant à adapter en fonction de la méthode de génération.

En conclusion, nous pouvons ajouter que dans un environnement réaliste, deux optimisations sont possibles. Premièrement, nous pouvons réduire l'espace de recherche des paires candidates. Pour ce faire, nous pouvons utiliser un algorithme de "local sensitive hashing". Nous avons testé la version utilisant des projections aléatoires pour générer le code de hachage. Sur les exemples que nous avons testés, les éléments quasi-dupliqués étaient toujours dans le même regroupement : [AI08, WSSJ14](choisir avec soin le nombre de bits et de hashtables). Ainsi, pour un nouveau jeu de données, nous ne testerons que les jeux de données dans le même regroupement que la nouvelle valeur. Les résultats peuvent être encore améliorés en utilisant une fonction de hachage basée sur la densité [JLLC13].

Deuxièmement, pour sélectionner plus finement les jeux de données identifiés comme presque dupliqués, nous pouvons calibrer [NMC05] les probabilités de sortie du classifieur et utiliser un seuil.

3.5.7 Conclusion

Dans cette section, nous avons introduit le concept de jeu de données quasi-dupliqués. Afin d'identifier ces jeux de données quasi-dupliqués, nous avons développé une méthode utilisant un classifieur pour distinguer les jeux de données quasi-dupliqués ne s'appuyant pas sur des méthodes existantes de comparaison de jeux de données. Pour tester cette méthode, nous avons

introduit une technique permettant de générer des exemples pertinents pour entraîner notre classifieur et le tester. Nous avons mené de nombreuses expériences afin d'identifier les meilleurs paramètres pour la méthode de génération. Toutes ces investigations nous ont permis d'obtenir un taux de reconnaissance de plus de 95%. Des recherches supplémentaires doivent être menées pour générer davantage de cas différents afin de rendre l'apprentissage plus robuste, tester avec un univers plus grand et tester d'autres métriques d'évaluation.

De plus, afin d'évaluer les résultats de notre algorithme dans d'autres conditions, il apparaît nécessaire de le tester sur des données générées à l'aide d'outils classiques de génération [CP09] et de pollution des données [CV13].

3.6 Conclusion du chapitre

Dans ce chapitre nous avons abordé de multiples problématiques relevant du profilage de données au sens large. Chacune de ces problématiques a été abordée à l'aide d'une même méthodologie exploitant un vecteur de métadonnées et de l'apprentissage automatique. Les pistes de recherche que nous avons explorées s'avèrent prometteuses mais les approches ont besoin d'être encore raffinées pour être industrialisables (utilisées dans l'industrie). Afin de faire un pas de plus dans cette direction et d'approfondir nos recherches en apprentissage automatique, nous allons dans le chapitre suivant développer nos travaux dans le domaine de la sélection de caractéristiques.

CHAPITRE 4

SÉLECTION DE CARACTÉRISTIQUES PAR ALGORITHME GÉNÉTIQUE

4.1 Introduction . . . 109
4.2 GAAM . . . 112
4.3 Ensemencement de population initiale . . . 114
4.3.1 Éclectique GA (EGA) . . . 114
4.3.2 Expériences . . . 115
4.3.2.1 Conditions d'expérience . . . 115
4.3.2.2 Techniques de création de la population initiale 116
4.3.3 Résultats . . . 118
4.3.3.1 Résultats avec l'algorithme EGA . . . 118
4.3.3.2 Résultats avec l'algorithme GAAM . . . 119
4.3.3.3 Comparaison avec des méthodes de réduction dimensionnelle . . . 122
4.3.4 Conclusion . . . 122
4.4 Échange entre la taille de la population et le taux de mutation pour GAAM . . . 123
4.4.1 Introduction . . . 123
4.4.2 mGAAM . . . 123
4.4.3 Expériences . . . 124
4.4.4 Résultats . . . 126
4.4.5 Test sur les vecteurs de meta-données . . . 128
4.4.6 Conclusion . . . 133
4.5 Conclusion du chapitre . . . 134

4.1 Introduction

Ce chapitre présente nos travaux dans le domaine de la sélection de caractéristiques à l'aide d'algorithmes génétiques. Les méthodes classiques pour la recherche de caractéristiques ont été détaillées dans la section 2.4.1. Non avons choisi d'utiliser des algorithmes génétiques pour la sélection de caractéristiques car ceux-ci présentent de meilleurs résultats que d'autres méthodes de type "wrapper". Pour faire ce choix, nous avons comparé l'algorithme génétique avec mutations agressives (GAAM présenté dans la section 4.2) à la méthode de sélection séquentielle de caractéristiques (SFS avant) sur 17 jeux de données décris dans le tableau 4.1.

TABLE 4.1 – Description des jeux de données utilisiés

Nom	Caract	Classes	Exemples	Source
Leaf [SMdS13]	15	30	340	openml.org/d/1482
Thoracic-surgery [ZTLS14]	16	2	470	openml.org/d/4329
Credit-g [Grö19]	20	2	1000	openml.org/d/31
Climate-model [LKT+13]	20	2	540	openml.org/d/40994
Dermatology	34	6	358	openml.org/d/35
Ionosphere	34	2	351	openml.org/d/59
Audit [HBR18]	36	2	1552	openml.org/d/42931
SPECTF	44	2	349	openml.org/d/1600
Hill-valley	100	2	1212	openml.org/d/1479
Spectrometer	101	48	531	openml.org/d/313
Musk	167	2	6598	openml.org/d/1116
Semeion	256	10	1593	openml.org/d/1501
Madelon [GGHD04]	500	2	2600	openml.org/d/1485
Har [AGO+13]	561	6	10299	openml.org/d/1478
Isolet	617	26	7797	openml.org/d/300
Parkinson-speech [SIS+13]	753	2	756	openml.org/d/42176
Micro-mass	1300	10	360	openml.org/d/1514

Dans chaque cas le taux de bonne reconnaissance est obtenu à l'aide d'une validation croisée 3 plis. Pour GAAM le résultat présenté est la valeur médiane calculée sur 50 simulations, en utilisant 10 individus et 100 générations, le classifieur utilisé est un classifieur bayesien naïf. Les résultats obtenus sont présentés dans les tableaux 4.2 et 4.3

TABLE 4.2 – Taux de bonne reconnaissance pour 10 caractéristiques, (validation croisée 3 plis) pour SFS et taux de bonne reconnaissance médian (50 simulations) pour GAAM part a

Données	Leaf	Thora	Credit	Clima	Derma	Ionos	Audit	SPEC
SFS	72.6	80.8	75.3	89.4	96.7	86.3	87.2	81.0
GAAM	**73.5**	**85.1**	**75.5**	**92.5**	**96.9**	**88.6**	**89.6**	**82.5**

Nous pouvons constater que les résultats que GAAM surpasse SFS dans 14 des 17 jeux de données. Il est important de noter que pour GAAM le nombre d'évaluations (validation croisée) est fixe et dans ce cas de 11010, tandis que pour SFS cela va dépendre du nombre total de caractéristiques n dans le jeu de données, ici $(10n - 45)$. Cependant à chaque évaluation SFS doit entraîner le modèle d'apprentissage automatique sur l'intégralité des ca-

TABLE 4.3 – Taux de bonne reconnaissance pour 10 caractéristiques, (validation croisée 3 plis) pour SFS et taux de bonne reconnaissance médian (50 simulations) pour GAAM part b

Données	Hill	Spect	Musk	Seme	Made	Har	Isol	Parki	Micro
SFS	**53.5**	53.8	95.2	62.7	63.4	88.2	73.2	**85.9**	**89.7**
GAAM	52.4	**57.7**	**95.3**	**66.9**	**63.9**	**91.0**	**75.0**	85.4	88.4

ractéristiques alors que GAAM sur seulement 10 (dans notre exemple). Ceci a une importance car la complexité de la plupart des modèles d'apprentissage automatique dépend directement du nombre de caractéristiques. Par exemple, si on a n caractéristiques et L le nombre d'exemples la complexité du classifieur Bayesien naïf est $\mathcal{O}(n * L)$.

Il est a noter qu'évaluer les modèles de la manière que nous avons présentée entraîne dans le cadre de la sélection de caractéristiques un biais sur les résultats [CT10]. Cependant, pour des usages pratiques ce biais peut être négligé [WC21](ce sera le cas dans la plupart des expériences sauf indication contraire). La méthode standard dans ce type de situation pour effectuer des test non biaisés est d'utiliser des validations croisées imbriquées. Dans notre cas, cela n'est pas possible du fait de la forte variance des résultats des algorithmes génétiques. Dans ce cas il est parfois conseillé de répéter les validations croisées puis de prendre la moyenne des résultats ; cependant cette méthode n'est pas fiable [VBDB$^+$01]. Nous avons donc choisi de générer des résultats non baisés en séparant cinquante fois les données en un ensemble d'apprentissage représentant 75% du jeux de données et un ensemble de test représentant 25% du jeu de données. Le résultat final correspondant à la moyenne des 50 évaluations sur les 50 jeux de test. Les résultats obtenus sont présentés dans les tableaux 4.4 et 4.5. On y constate que les résultats obtenus sont généralement plus faibles que précédemment (dû au biais) mais aussi que GAAM surpasse SFS dans 12 des 17 jeux de données et qu'ainsi les résultats de la comparaison des deux algorithmes ne change que peu.

TABLE 4.4 – Taux de bonne reconnaissance moyen pour 10 caractéristiques, obtenu en répétant 50 fois une séparation 75% apprentissage 25% test part a

Données	Leaf	Thora	Credit	Clima	Derma	Ionos	Audit	SPEC
SFS	70.3	78.6	72.1	**90.6**	93.5	**91**	92.8	75
GAAM	**71.4**	**84.2**	**72.2**	90.5	**93.8**	90.2	**93**	**76.1**

SFS présente l'avantage d'être déterministe et plus rapide si l'on a un

TABLE 4.5 – Taux de bonne reconnaissance moyen pour 10 caractéristiques, obtenu en répétant 50 fois une séparation 75% apprentissage 25% test part b

Données	Hill	Spect	Musk	Seme	Made	Har	Isol	Parki	Micro
SFS	50	48.6	99.4	58	**60.7**	91	73.6	80.2	85.2
GAAM	50	**50.2**	**99.5**	**63**	60.5	91	**73.9**	**80.9**	**86**

faible nombre de caractéristiques totales. Cependant, il ne permet pas d'obtenir les meilleurs résultats possibles. Ainsi, dans tout ce chapitre, on s'intéressera particulièrement à l'algorithme GAAM.

GAAM a été développé en réponse aux problèmes des algorithmes génétiques classiques dans le cadre de la sélection de caractéristiques. En effet, de part leur fonctionnement (initialisation aléatoire donc en moyenne 50% de 0 et 50% de 1) et l'espace de recherche très rugueux les algorithmes génétiques basés sur une fonction mono objectif simple tendent à converger dans un optimum local (utilisant environ 50% des caractéristiques). Ce problème persiste même en cas de transformation de la fonction à optimiser en une fonction plus complexe prenant en compte le nombre de caractéristiques utilisées (en scalarisant linéairement le problème multi-objectif) [RL15].

4.2 GAAM

L'algorithme génétique avec mutations agressives (GAAM) est un algorithme qui a été spécifiquement conçu pour la sélection de caractéristiques [Rej14]. Il est spécialisé pour la sélection d'un faible nombre de caractéristiques. Cet algorithme présente plusieurs différences majeures par rapport aux algorithmes génétiques classiques. Tout d'abord, l'encodage des gènes n'utilise pas un système binaire mais des entiers. Si un jeu de données contient N caractéristiques, l'encodage utilise N entiers, chaque nombre représentant une caractéristique. De plus, la valeur 0 représente l'absence de caractéristiques. Ensuite l'algorithme permet de choisir le nombre maximum de caractéristiques à conserver, ce nombre correspond à la taille du chromosome choisi. Par exemple, si un jeu de données a 10 caractéristiques et que nous voulons en garder un maximum de 5, des exemples de chromosomes seraient : [2 7 9 10 8], [3 2 2 10 5] [0 7 8 9 5]. Si un gène est présent plusieurs fois comme dans [3 2 2 10 5], les doublons sont éliminés et remplacés par 0. Ensuite le système de mutation est différent de celui des algorithmes traditionnels. En effet, chaque individu va permettre de créer un nombre de mutants égal à sa taille. Chaque mutant est une copie du chromosome initial

où seul un gène a été muté (par tirage aléatoire uniforme entre 0 et la taille du chromosome), pour le premier individu mutant c'est le premier gène qui est muté, pour le second le second gène et ainsi de suite. Si nous avons le gène [2 3 4] les mutants seront par exemple [4 3 4], [2 0 4] et [2 3 2]. Le croisement est un croisement classique à un point, chaque individu est croisé afin de générer une population fille de taille égale à la taille de la population initiale. Ainsi, l'algorithme ne nécessite pas de paramètre représentant la probabilité de mutation ou la probabilité de croisement.

La dernière différence entre l'algorithme GAAM et les algorithmes génétiques classiques réside dans l'ordre dans lequel les opérations sont effectuées. En effet, la population initiale générée aléatoirement n'est pas évaluée au début de l'algorithme, elle est simplement utilisée pour générer la population fille et la population mutante. Ainsi, si l'on part de N individus avec des chromosomes de taille T, on obtient N individus dans la population fille et N*T individus dans la population mutante. Nous avons donc 2*N + N*T individus à évaluer à la fin de la première génération. On garde alors les N meilleurs individus. L'algorithme est illustré dans la figure 4.1. Si l'on note G le nombre de générations, la complexité de l'algorithme est de l'ordre de $\mathcal{O}(N * G * T * \mathcal{O}(classifieur))$.

Algorithm 10 GAAM, G représente le nombre de générations, N le nombre d'individus, T la taille des chromosomes, L le nombre de colonnes dans le jeux de données

INPUT : *G :int, N :int, T :int ,L :int*
0 $g = 0$
1 : Construire N individus avec T gènes choisis aléatoirement dans {0,1,2..L} dans le but de créer la population initiale *Ip*
2 : Agressive mutation : Créer *Mp* la population mutée
for j=1 to N **do**
  for x=1 to T **do**
    Choisir une valeur aléatoire *m* dans *{0,1,2..L}*
    Assigner à *D* une copie de **Ip(j)**
    D(x)=m
    Ajouter *D* à *Mp*
  end for
end for
3 Croisement : appliquer un classique croisement de holland à chaque individu
4 : Créer *Tp=Ip+Mp+Cp* évaluer chaque individu de la population totale et les classer par le résultat de leur fonction d'évaluation.
5 : Supprimer les *N+N*T* individus avec les plus faibles résultats d'évaluation de *Tp* et remplacer *Ip* par les individus restants, *g+=1*
6 Si $g = G$ retourner *Ip(0)* sinon retour à l'étape 2.

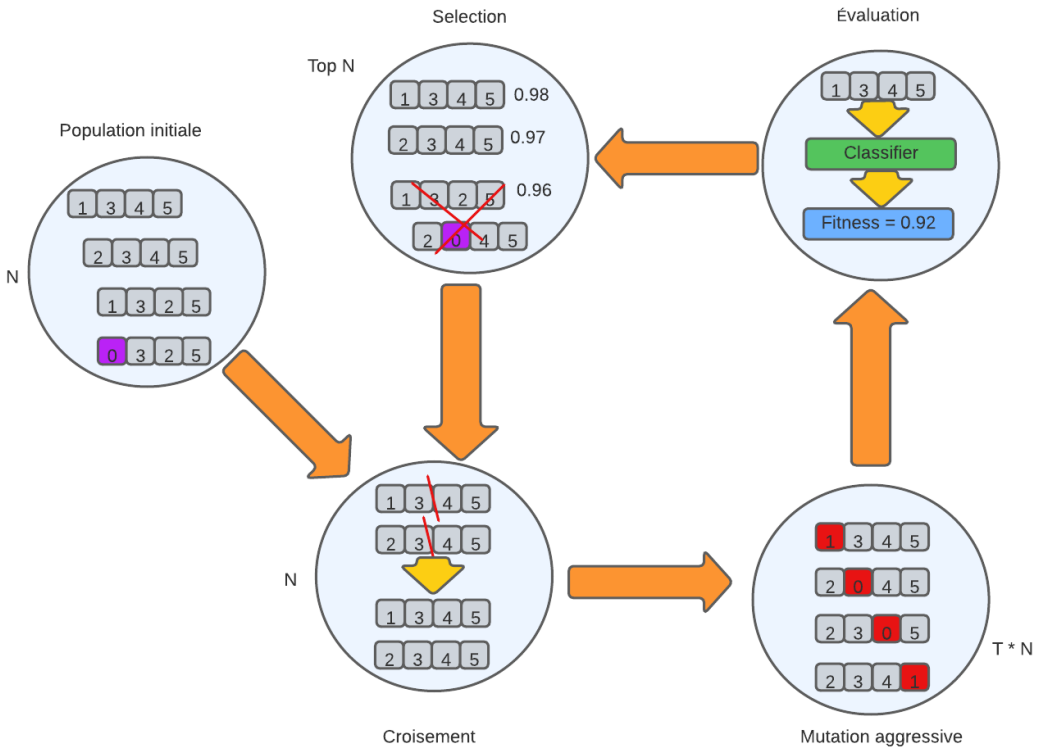


FIGURE 4.1 – Illustration du fonctionnement de l'algorithme GAAM

4.3 Ensemencement de population initiale

Dans certains cas d'usage des algorithmes génétiques, initialiser la population de départ avec des individus non aléatoires permet d'obtenir de meilleurs résultats [OCD+14]. Le principe est de chercher de bonnes solutions avec un autre processus d'optimisation, d'encoder ces solutions puis de les injecter dans la population initiale de l'AG. Nous avons donc décidé d'appliquer ce type de méthode au cas de la sélection de caractéristiques. Cette section présentera les résultats de nos premières investigations dans ce domaine.

4.3.1 Éclectique GA (EGA)

Nous allons tout d'abord introduire l'algorithme génétique classique appelé Éclectique GA (EGA) qui sera utilisé dans les tests. EGA utilise un croisement annulaire, des mutations uniformes ainsi qu'un élitisme total [KMAB13].

Nous avons choisi cet algorithme comme exemple d'AG généraliste car il présente dans plusieurs tests de meilleurs résultats que d'autres possibilités d'AG.

Si l'on note G le nombre de générations, n le nombre d'individus, *B2M* le nombre de gènes à muter, *I(n)* le *n-ième* individu, L la longueur d'un

chromosome, P_c la probabilité de croisement et P_m la probabilité de mutation.

EGA [KMAB13], est décrit dans l'algorithme 11.

Algorithm 11 EGA

INPUT : *G :int , n :int, L :int*
Étape 0 $B2M \leftarrow [n * L * P_m]$
Étape 1 $i \leftarrow 1$
Étape 2 Générer une population aléatoire
Étape 3 Évaluer la population
for $j \leftarrow 1$ **to** n **do**
  $I(n+j) \leftarrow I(j)$
  $fitness(n+j) \leftarrow fitness(j)$
end
Étape 5 Croisement annulaire
Étape 6 Mutation
for $j \leftarrow 1$ **to** $B2M$ **do**
  Générer des nombres aléatoires de manière uniforme $0 < \rho_1 < 1$ et $0 < \rho_2 < 1$
  Muter le bit round(ρ_2*L) de I[round(ρ_1*n)]
end
Étape 7 Sélection
Classer les 2n individus par leur fitness, de manière ascendante
Étape 8 $i \leftarrow i + 1$
if $i = G$ **then**
  OUTPUT I(1) et arrêt
else
  Aller à Étape 3
end

4.3.2 Expériences

4.3.2.1 Conditions d'expérience

Tous nos tests ont été effectués sur des instances google colab, équipées de processeurs Intel Xeon 2.30GHz à 4 coeurs et 12Go de RAM.

Pour nos expériences, nous avons choisi cinq jeux de données parmi ceux présentés au début du chapitre ils sont décrits dans le tableau 4.6.

Nous avons concentré notre analyse sur deux jeux de données Semeion et Madelon. Nous avons comparé différents niveaux d'ensemencement de la population initiale : 0%, 10% , 50% et 100%. Pour EGA, nous avons sélectionné une population initiale de 100 individus, et choisi comme maximum d'itérations 100. Les probabilités de mutation et de croisement sont calculées dynamiquement en suivant la stratégie ILM/DHC [HAA+19] :

TABLE 4.6 – Description des jeux de données testées pour l'ensemencement d'AG

Nom	Caract	Exemples	Classes
Madelon[GGHD04]	500	2600	2
Semeion	256	1593	10
Har[AGO+13]	561	10299	6
Parkinson-speech [SIS+13]	753	754	2
Hill-Valley	100	1212	2

On note Gi le numéro de la génération courante, Gm le nombre maximum de générations.

$$P_c = 1 - \frac{Gi}{Gm} \tag{4.1}$$

$$P_m = \frac{Gi}{Gm} \tag{4.2}$$

L'avantage d'utiliser cette méthode, est qu'elle permet d'avoir un fort taux de croisement au départ, cela va favoriser une meilleure exploration de l'espace de recherche afin de trouver un bon optimum. Celui-ci va ensuite décroître en faveur d'un fort taux de mutation qui va permettre une plus grande possibilité d'optimisation des individus.

Le classifieur utilisé est un classifieur bayésien naïf. Les résultats sont mesurés à partir de la moyenne du taux de bonne reconnaissance calculé à l'aide d'une validation croisée 5 plis et est noté *ma*. La fonction d'évaluation (fitness) est définie par 1 - *ma*. Pour l'algorithme GAAM, nous avons décidé de conserver 20 caractéristiques, d'utiliser une population de 20 individus, d'utiliser la même fonction d'évaluation ainsi que de nous limiter à 100 itérations.

4.3.2.2 Techniques de création de la population initiale

Pour l'initialisation de la population, nous avons développé deux méthodes notées SSM (Standard Seeding Method) et ESM (Elitism Seeding Method). Notre modèle pour la création des chromosomes nécessaires à l'initialisation s'appuie sur l'utilisation d'une forêt d'arbres décisionnels [Bre01]. Nous entraînons tout d'abord une RF sur un échantillon du jeu de données traitées avec les paramètres suivants : 100 estimateurs, profondeur maximale 8. L'importance de chaque caractéristique est évaluée par la mesure d'impureté Gini et normalisée (la somme du score de toutes les caractéristiques

est égale à 1). Ensuite, nous utilisons cette évaluation comme probabilité de sélection pour chaque caractéristique dans notre algorithme.

Pour SSM, nous utilisons les probabilités préalablement définies pour sélectionner aléatoirement le nombre de caractéristiques pour chaque individu.

Pour ESM nous procédons de la même manière mais en générant cinq fois plus d'individus que nécessaire puis nous évaluons ces individus à l'aide d'un classifieur bayésien naïf (sur le même échantillon qui a servis à entraîner la RF) et nous ne gardons que les N premiers individus dont nous avons besoin.

Les chromosomes générés de cette façon peuvent avoir des valeurs dupliquées. GAAM gère nativement ce problème en ne gardant qu'une des valeurs. Pour EGA, lors de la conversion en binaire des gènes, les gènes dupliqués sont supprimés.

Pour s'assurer de la validité de la méthode, nous avons généré 1000 exemples en utilisant le jeu de données Semeion, puis nous avons utilisé un classifieur bayésien pour les évaluer. Le taux de bonne reconnaissance est calculé à l'aide d'une validation croisée 5 plis. Pour chaque méthode, les résultats sont présentés dans la Fig 4.2.

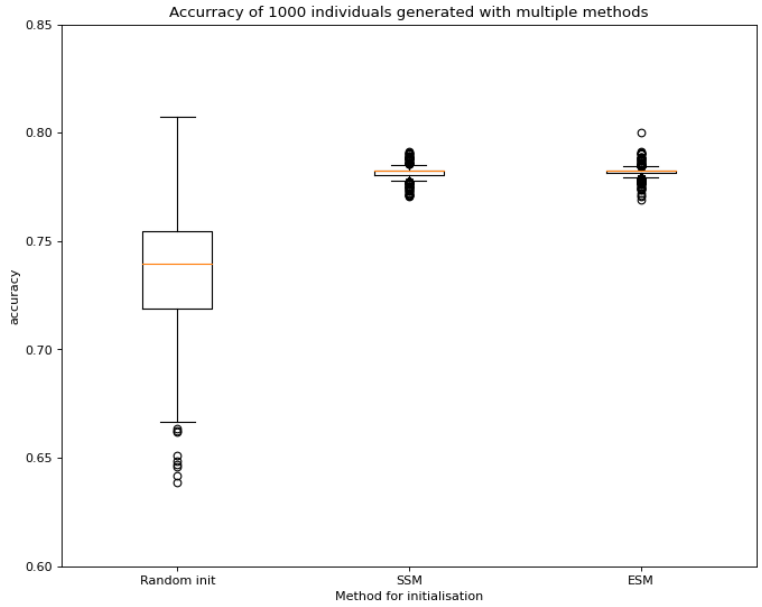


FIGURE 4.2 – Boite à moustache du taux de reconnaissance pour 1000 individus générés avec chaque méthode

Le taux de reconnaissance médian des groupes générés à l'aide de SSM et de ESM est plus élevé que ceux générés à l'aide de la méthode standard.

Ces résultats indiquent que les deux méthodes génèrent des individus initiaux meilleurs que ceux générés aléatoirement (cependant de meilleurs individus initiaux ne garantissent pas une meilleure population finale). Cette information est particulièrement intéressante pour SSM, les individus initiaux n'étant pas pré-sélectionnés comme avec ESM, rien ne garantissait qu'ils soient meilleurs que la moyenne. De plus les individus générés présentent

beaucoup moins de variabilité au niveau des résultats tout en gardant une assez grande diversité. Il est à noter que la méthode SSM permet de garder une meilleure diversité que la méthode ESM.

4.3.3 Résultats

Du fait de la nature probabiliste des algorithmes génétiques, chaque simulation est effectuée 10 fois et la moyenne des 10 simulations est conservée. Nous pouvons tirer quelques conclusions préliminaires de ces résultats.

Sur les 60 expériences que nous avons menées en utilisant nos techniques d'initialisation, 50 présentent de meilleurs résultats que celles sans initialisation. Pour les 10 résultats restants, ils sont quasiment identiques à ceux sans technique d'initialisation. Cela confirme l'utilité de l'approche. Ces résultats sont logiques car les individus qui sont injectés dans la population initiale via nos méthodes sont déjà passés à travers un processus d'optimisation avec le démarrage de l'AG.

Dans la section suivante, nous allons analyser plus profondément les résultats sur Semeion et Madelon.

4.3.3.1 Résultats avec l'algorithme EGA

Pour EGA, les résultats sont particulièrement intéressants car l'initialisation par nos techniques mène majoritairement à un meilleur taux de reconnaissance et une meilleure stabilité des résultats. Les Fig.4.3 et 4.4 présentent les résultats de EGA en utilisant la technique SSM.

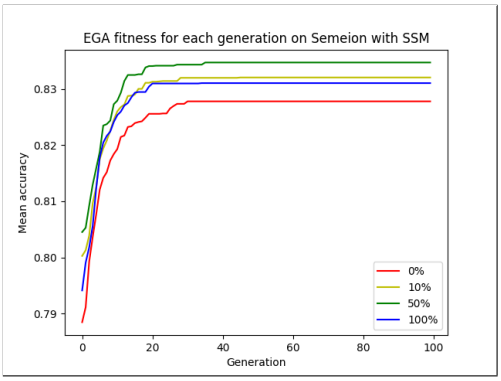


FIGURE 4.3 – Évolution du taux de bonne reconnaissance moyen calculé sur 10 simulations en utilisant SSM sur Semeion

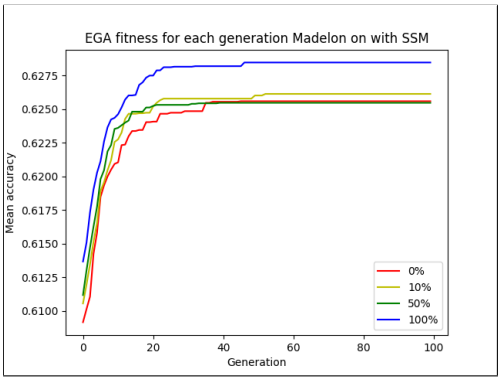


FIGURE 4.4 – Évolution du taux de bonne reconnaissance moyen calculé sur 10 simulations en utilisant SSM sur Madelon

Les résultats indiquent que notre technique permet d'obtenir de meilleurs résultats qu'une initialisation aléatoire. De plus, dans le pire scénario, les résultats sont équivalents à ceux réalisés par la méthode d'initialisation aléatoire. La population issue de notre technique étant déjà plus optimale que la population issue d'une initialisation aléatoire, elle présente toujours de

meilleurs résultats jusqu'à la vingtième génération. La vitesse de convergence n'est pas affectée par notre technique. De plus, les résultats d'EGA se stabilisent toujours après environ 50 générations. Notre technique nous permet d'atteindre des taux de reconnaissance plus élevés.

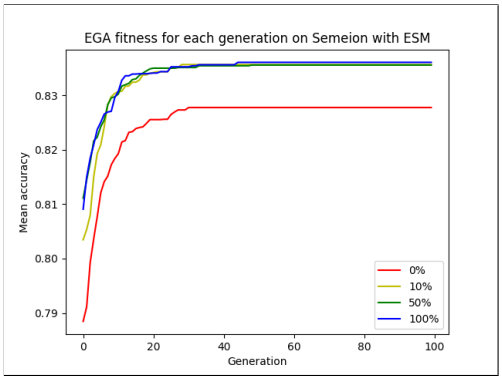


FIGURE 4.5 – Évolution du taux de bonne reconnaissance moyen calculé sur 10 simulations en utilisant ESM sur Semeion

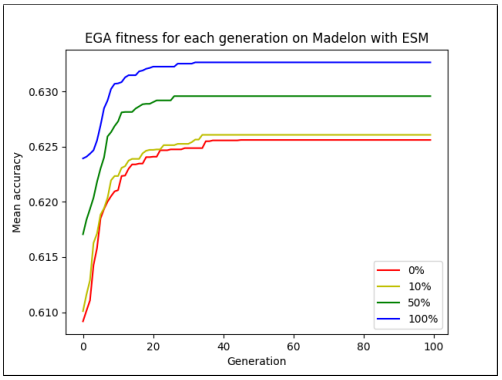


FIGURE 4.6 – Évolution du taux de bonne reconnaissance moyen calculé sur 10 simulations en utilisant ESM sur Madelon

La technique ESM (Fig 4.5 et 4.6), performe toujours mieux qu'une initialisation aléatoire. Sur Semeion qui ne contient que 256 caractéristiques, qu'importe le taux d'ensemencement l'algorithme converge toujours vers le même résultat. En revanche, sur Madelon, plus le taux d'ensemencement est grand meilleurs sont les résultats. On peut en conclure que ESM renvoie toujours un groupe d'individus initiaux qui sont bons, ce qui conduit l'algorithme à converger vers un optimum meilleur qu'avec une initialisation aléatoire.

Suite à ces résultats, nous avons ré-effectué certains tests avec 50 simulations. Il en ressort globalement que ESM et SSM présentent de meilleurs résultats que la méthode d'initialisation standard. Cependant l'usage de la méthode SSM est à privilégier. En effet ESM, en plus de présenter un surcoût en calcul génère des résultats avec une plus grande variance. Cela est probablement la conséquence d'une trop faible diversité génétique dans la population initiale, cette trop faible diversité entraînant la convergence de l'algorithme dans un optimum local trop rapidement.

4.3.3.2 Résultats avec l'algorithme GAAM

Avec l'algorithme GAAM (Fig 4.7, 4.8 ,4.9 et 4.10), la taille de l'espace de recherche est fortement réduite car le nombre de caractéristiques à sélectionner est fixé dès le départ. Cette réduction mène à moins de possibilités d'optimisation. Comme prévu, les résultats sont meilleurs sur Madelon que sur Semeion car l'espace de recherche est plus grand. Avec SSM, une proportion de 10% d'individus optimisés dans la population initiale mène aux

meilleurs résultats dans tous les cas. Ce résultat peut être expliqué par le fonctionnement de GAAM. Si à cause de notre méthode les individus initiaux sont trop similaires (gros taux d'ensemencement) l'algorithme va vite converger vers un optimum local dont il aura du mal à sortir du fait de la méthode de mutation. Les possibilités d'exploration de l'espace des solutions sont ainsi bridés.

Nous avons comme précédemment renouvelé certains tests avec 50 simulations. Nous pouvons en conclure que les résultats en matière de taux de bonne reconnaissance, il n'y a pas de différence significative entres les 3 méthodes. L'avantage principal d'utiliser SSM est une réduction de la variance des résultats par rapport à la méthode standard.

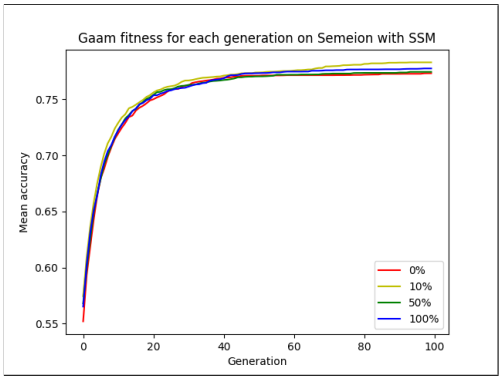


FIGURE 4.7 – Évolution du taux de bonne reconnaissance moyen calculé sur 10 simulations en utilisant SSM sur Semeion

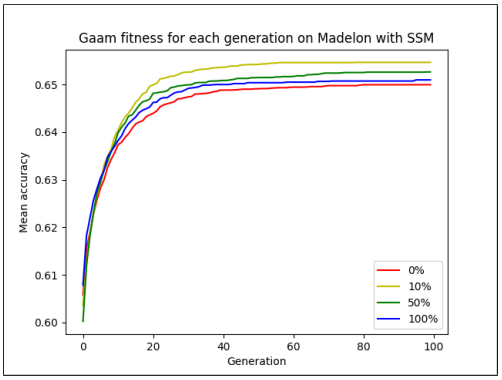


FIGURE 4.8 – Évolution du taux de bonne reconnaissance moyen calculé sur 10 simulations en utilisant SSM sur Madelon

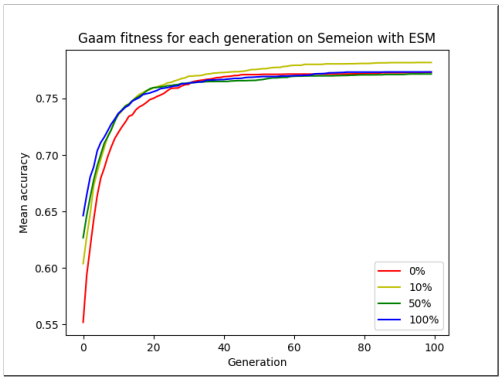


FIGURE 4.9 – Évolution du taux de bonne reconnaissance moyen calculé sur 10 simulations en utilisant ESM sur Semeion

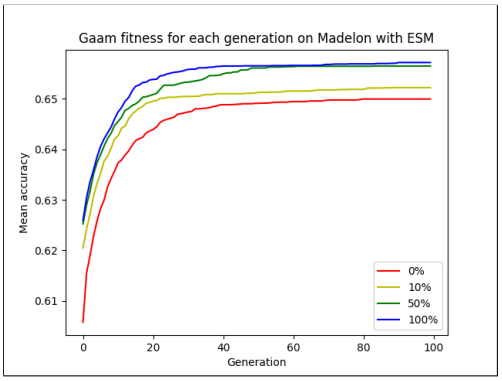


FIGURE 4.10 – Évolution du taux de bonne reconnaissance moyen calculé sur 10 simulations en utilisant ESM sur Madelon

Les résultats de nos expériences sont résumés dans la Table 4.7 et la Table 4.8. Nous avons ajouté l'écart type calculé sur les 10 simulations pour évaluer la dispersion des résultats pour chaque scénario.

TABLE 4.7 – Taux de bonne reconnaissance (Tbr) pour le jeu de données Madelon, 10 simulations

Madelon	Standard	SSM			ESM		
% Ensemencement	0%	10%	50%	100%	10%	50%	100%
GAAM Tbr	.649	.655	.653	.651	.652	.656	.657
std	.007	.004	.005	.002	.007	.008	.004
Ega Tbr	.625	.626	.625	.628	.626	.630	.632
std	.002	.001	.001	.001	.002	.0018	.001

TABLE 4.8 – Taux de bonne reconnaissance (Tbr) pour le jeu de données Semeion, 10 simulations

Semeion	Standard	SSM			ESM		
% Ensemencement	0%	10%	50%	100%	10%	50%	100%
GAAM Tbr	.773	.783	.774	.777	.781	.771	.773
std	.002	.007	.009	.008	.004	.009	.006
Ega Tbr	.827	.832	.834	.831	.835	.835	.836
std	.0028	.0027	.0024	.0026	.0041	.0027	.0013

Enfin pour confirmer les résultats nous avons testé les meilleurs niveaux d'ensemencement pour les deux méthodes sur Madelon et Semeion en utilisant la méthodologie de test (non biaisée) décrite dans la section 4.1. Les résultats sont dans le tableau 4.9. On y constate une légère amélioration des résultats quand les techniques d'ensemencement sont utilisées, l'avantage est toujours plus important sur l'algorithme EGA.

TABLE 4.9 – Taux de bonne reconnaissance moyen pour EGA et GAAM 20 caractéristiques, pour les deux techniques d'ensemencement, obtenu en répétant 50 fois une séparation 75% apprentissage 25%

Données	Sem	Mad
EGA	80.9	59.9
EGA SSM 100%	81.2	59.9
EGA ESM 100%	**81.4**	**60.1**
GAAM 20	73.5	60.4
GAAM 20 SSM 10%	**73.8**	**60.6**
GAAM 20 ESM 10%	73	**60.6**

4.3.3.3 Comparaison avec des méthodes de réduction dimensionnelle

Nous avons comparé les résultats des deux algorithmes génétiques (de sélection dimensionnelles) améliorés par notre méthode avec deux méthodes classiques de réduction dimensionnelle : l'ACP [F.R01] et les réseaux de neurones auto-encodeur [WYZ16] sur les cinq jeux de données.

Ces tests sont effectués avec deux paramétrages différents. Le premier est une réduction dimensionnelle à 20 dimensions afin de pouvoir comparer les résultats avec ceux de GAAM. Le deuxième est une réduction dimensionnelle à une taille égale à la moitié du nombre de caractéristiques initiales afin de pouvoir comparer avec EGA (les algorithmes génétiques classiques utilisant comme critère d'évaluation le taux de bonne reconnaissance tendent à converger vers ce résultat [RL15].

Les résultats (dans les tableaux 4.10 et 4.11) indiquent que les AG utilisant nos méthodes ont de meilleurs résultats que les méthodes de réduction dimensionnelle sur quatre jeux de données sur cinq, les résultats sont particulièrement bon si le jeu de données contient plus de 500 caractéristiques.

TABLE 4.10 – Résultats avec 20 caractéristiques

Methodes | Données	Madel	Semei	Hill-Vall	Har	Parkin
PCA 20 caract	.627	**.867**	.511	.794	.725
Auto-encoder 20 caract	.550	.784	.511	.779	.737
GAAM SSM meilleur résultat	.654	.783	.518	**.927**	.871
GAAM ESM meilleur résultat	**.657**	.781	**.519**	.927	**.873**

TABLE 4.11 – Résultats sur la moitié du nombre de caractéristiques

Methodes | Données	Madel	Seme	Hill-Vall	Har	Parkin
PCA	.582	**.883**	.504	.675	.681
Auto-encodeur	.545	.819	.510	.779	.7369
EGA SSM meilleur résultat	.628	.835	.518	.878	.805
EGA ESM meilleur résultat	**.632**	.836	**.519**	**.881**	**.807**

4.3.4 Conclusion

Dans cette section, nous avons étudié l'influence de méthodes d'ensemencement pour les algorithmes génétiques dans le cadre de la sélection de caractéristiques [CRB+21]. Les expériences ont été réalisées en utilisant deux

algorithmes génétiques, deux méthodes d'initialisation et cinq jeux de données. Dans chaque scénario, trois taux d'ensemencement ont été comparés. Nos méthodes ont aussi été comparées à deux algorithmes classiques de réduction dimensionnelle.
Nos résultats indiquent qu'initialiser de manière non aléatoire au moins une partie de la population de départ des AG est une méthode utile pour améliorer leurs résultats dans le cadre de la sélection de caractéristiques, principalement pour les jeux de données sont de grande dimension. Ces résultats sont prometteurs, et laisse entrevoir des possibilités d'application sur les vecteurs que nous avons présentés dans les chapitres précédents. Cependant, d'autres études doivent être menées, tout d'abord pour élaborer une méthode fiable pour déterminer le taux d'ensemencement optimal à utiliser sans avoir à tester tous les cas. Ensuite il serait intéressant d'analyser l'effet d'utiliser plusieurs sources d'individus optimisés au lieu d'une seule.

4.4 Échange entre la taille de la population et le taux de mutation pour GAAM

4.4.1 Introduction

L'algorithme GAAM que nous avons présenté dans la section 4.2 est un outil puissant pour la sélection de caractéristiques, il brille principalement quand le nombre de caractéristiques à conserver est faible. Lors de nos expérimentations avec celui-ci, il nous est apparu qu'il présentait plusieurs axes d'amélioration possibles. Nous allons dans les sections suivantes décrire nos recherches sur l'un de ces axes d'amélioration.

4.4.2 mGAAM

L'algorithme de GAAM est très efficace mais le principe de mutation agressive oblige à n'utiliser qu'un petit nombre d'individus initiaux afin de ne pas avoir trop d'individus à évaluer après l'étape de mutation. Ce faible nombre d'individus initiaux va conduire à une faible diversité génétique au sein de la population, ce qui peut amener l'algorithme à rester piégé dans un optimum local. Nous avons voulu évaluer l'impact de la taille de la population initiale sur les résultats de l'algorithme. Nous avons donc défini une version de GAAM, nommée mGAAM [CGB+22]. La méthode se définie en comparaison de l'algorithme GAAM classique. Ainsi notre algorithme prend comme paramètre la taille d'une population hypothétique de comparaison qui serait utilisée avec la version standard de GAAM.

Notre objectif est de garder globalement le même nombre d'individus créés ainsi qu'un nombre d'individus générés à chaque itération similaire qu'une version hypothétique de GAAM avec une population de taille donnée ; ceci tout en proposant un nombre plus faible ou plus élevé d'individus dans la

population initiale. De plus nous conserverons le même nombre d'individus mutés que dans la version originale de comparaison.
Ainsi, le paramètre qui sera ajusté est le nombre de chromosomes mutants générés par un chromosome lors de l'étape de mutation. Si nous utilisons une population plus grande que la version initiale, par exemple 20 au lieu de 10, le nombre d'individus mutés sera réduit afin d'obtenir en moyenne le même nombre d'individus par itération que la version originale de l'algorithme.

Notre méthode modifie les étapes 2 et 3 de l'algorithme GAAM. Pour l'étape 3, nous ne gardons qu'un nombre d'individus croisés égal à gaam_pop_size. Ce nombre correspond à la taille de la population à laquelle nous nous comparons. On applique l'opération de croisement à tous les individus puis sélectionne aléatoirement le nombre désiré. Nous ne cherchons pas à avoir un grand nombre d'individus croisés car ils n'ont que peu d'influence dans les résultats de l'algorithme.
L'algorithme 12 remplace l'étape de mutation de la version originale. Il suffit d'ajouter à la version originale un paramètre gaam_pop_size qui décrit la taille de la population hypothétique à laquelle on souhaite se comparer. De plus, le N correspond à mgaam_pop_size dans l'algorithme 12. Lorsque la taille de la population choisie pour mGAAM est inférieure à la taille de la population cible hypothétique, chaque gène est possiblement muté plusieurs fois. Cependant cela mène quasi-systématiquement à une convergence très rapide de l'algorithme vers un optimum local.

L'ensemble du processus de mutation étant probabiliste le nombre d'individus générés à chaque génération n'est pas fixe. Cependant, la taille moyenne (calculée sur l'ensemble des générations) de la population à chaque itération (avant l'étape de sélection) correspond à la taille de celle de l'algorithme GAAM classique (pour les paramètres que nous avons choisis).

4.4.3 Expériences

Pour effectuer les calculs, nous utilisons une instance de Google Colab [Bis19] avec un Xeon 2.30GHz 4-core et 25Go Ram. Le classifieur utilisé est un classifieur bayésien naïf, nous avons choisi ce classifieur car il a l'avantage d'être extrêmement rapide à entraîner. L'évaluation des résultats se fait en utilisant la moyenne des taux de bonne reconnaissance calculés à l'aide de la validation croisée 3 plis.
Les 17 jeux de données utilisés sont présents et décrits dans le référentiel d'apprentissage automatique de l'UCI [DG17]. Nous avons utilisé les versions de ces jeux de données librement disponibles sur Open ML [VvRBT14, FvRK$^+$19]. Les jeux de données utilisés sont décrits dans le tableau 4.1. Les jeux de données contiennent principalement des données numériques, les données catégorielles sont encodées numériquement. Chaque résultat présenté

Algorithm 12 Fonction Mutation de mGaam pop la population, L le nombre de caractéristiques dans le jeu de données, mgaam_pop_size : taille de la population, gaam_pop_size taille de la population de comparaison, T taille du chromosome

INPUT : *pop :list, L :int, mgaam_pop_size :int, gaam_pop_size :int, T :int*

INIT : tm ← Calc_tm_standard(mgaam_pop_size, gaam_pop_size,T)
offspring ← []

for *j=1 to size(pop)* **do**
  tmp_offspring ← []
  tmp_tm = copy(tm)
  while *tmp_tm>0* **do**
    for *i=1 to T* **do**
      ind *gets* copy(pop[j])
      $rho \leftarrow random\_uniform(0, 1)$
      if $rho < tmp\_tm$ **then**
        new_val ← random value between 0 and L
      ind[i]← new_val
      add ind to tmp_offspring
    tmp_tm ← tmp_tm - 1
  offspring ← offspring + tmp_offspring
OUTPUT offspring

Algorithm 13 : **Calc_tm_standard** mgaam_pop_size : taille de la population, gaam_pop_size taille de la population de comparaison, T taille du chromosome

INPUT : *mgaam_pop_size :int, gaam_pop_size :int, T :int*

$\text{tm} \leftarrow \frac{gaam\_pop\_size*(T+1)-mgaam\_pop\_size)}{T*mgaam\_pop\_size}$
OUTPUT *tm*

est calculé à partir de 50 simulations. Le taux de mutation pour mGAAM sont calculés automatiquement pour que le nombre d'individus reste égal (à chaque génération) à celui qui serait utilisé par l'algorithme GAAM original hypothétique avec 10 individus et 100 itérations.

4.4.4 Résultats

Les résultats de nos expériences sont présentés dans le tableau 4.13,4.14,4.15 et 4.16. La colonne correspondant à l'algorithme original est celle où la taille de la population est de 10. On remarque que dans la majorité des jeux de données, les résultats sont meilleurs lorsque la taille de la population est supérieure à 10, très souvent une taille de 60 (donc avec un faible taux de mutation) donne de meilleurs résultats. Une explication possible est le fait que l'utilisation d'une population de départ plus importante permet d'obtenir en moyenne de meilleurs individus à la fin de la première itération ainsi qu'une plus grande diversité génétique (comme nous conservons plus d'individus d'une génération à l'autre).

Ces individus initiaux meilleurs et plus diversifiés limite la convergence trop rapide de l'algorithme dans un optimum local et permettent ensuite d'obtenir de meilleurs résultats à la fin des 100 générations. Cet effet peut être observé dans la figure 4.11 qui représente les résultats de taux de bonne reconnaissance moyen à chaque itération pour le jeu de données Credit g. On peut également constater sur la figure 4.11 qu'il est inutile d'augmenter indéfiniment la population, la population de taille 90 n'ayant aucun avantage sur la population de taille 60.

Nous pouvons constater que progressivement, lorsque le nombre de caractéristiques augmente dans les jeux de données, les meilleures solutions sont trouvées pour des tailles de population plus petites. Ceci s'explique par le fait que notre méthode essaie d'avoir à chaque itération le même nombre d'individus créés que l'algorithme initial. Cette contrainte implique qu'à chaque génération, lorsque la population est supérieure à 10, nous évaluons moins de nouveaux individus que l'algorithme GAAM initial. En effet nous ne réévaluons pas les individus déjà évalués et conservés à la génération suivante dans un but d'optimisation.

Par exemple l'algorithme original et les paramètres de l'expérience, à chaque itération (sauf la première), 10 individus issus de croisements et 100 individus issus de mutations sont évalués. Alors qu'avec la méthode que nous avons utilisée dans l'expérience, si la population est de 30, il n'y a en moyenne que 90 nouveaux individus évalués à chaque génération, 10 issus de croisement, en moyenne 80 issus de mutation (car notre méthode cherche à générer

un nombre d'individus créés équivalent à celui de la version hypothétique et non le nombre d'individus évalués).

Ceci a pour effet de rendre l'algorithme plus rapide mais réduit également la vitesse de convergence. Mais lorsque le nombre de caractéristiques devient important, l'algorithme n'a pas le temps de converger complètement car le problème devient plus difficile. Cet effet peut être visualisé dans la figure 4.12 sur le jeu de données Micro-mass.

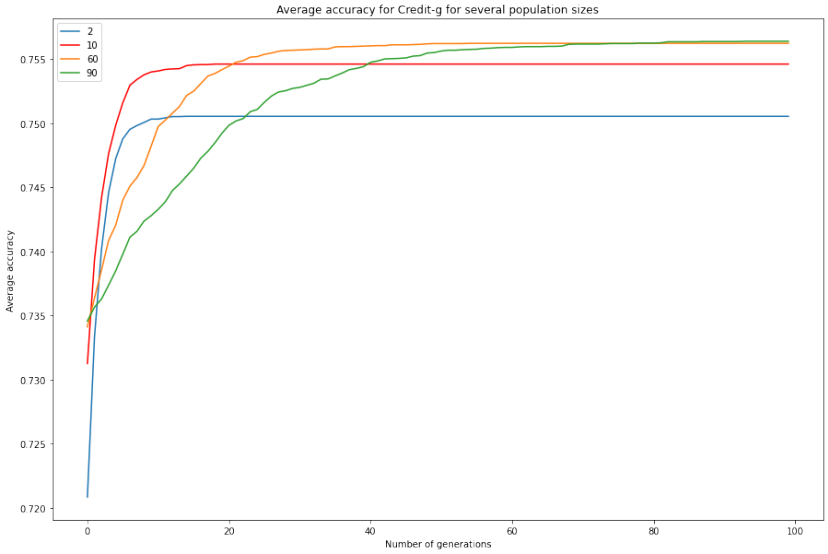


FIGURE 4.11 – Taux de bonne reconnaissance moyen sur Leaf pour plusieurs tailles de population

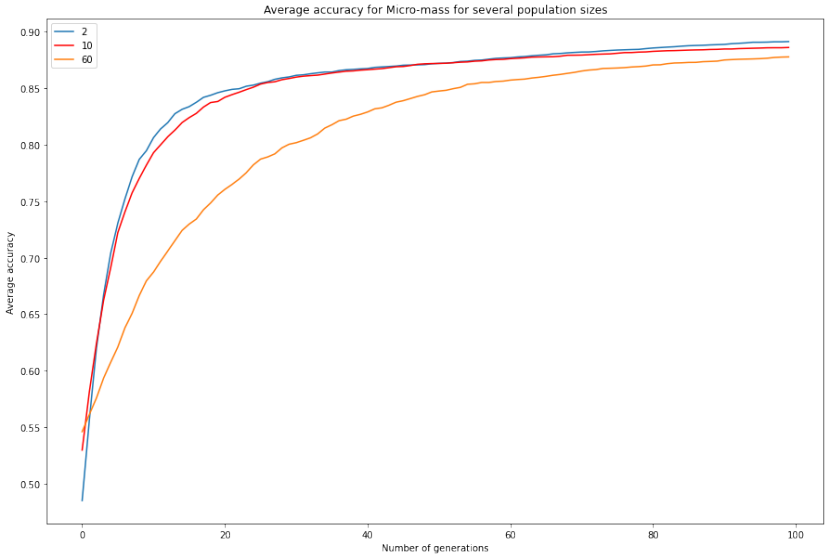


FIGURE 4.12 – Taux de bonne reconnaissance moyen sur Micro-Mass pour plusieurs tailles de population

Ce problème est supprimé en remplaçant l'algorithme de calcul du taux

de mutation par l'algorithme 14. Celui-ci permet d'obtenir un plus grand nombre d'individus évalués à chaque itération (équivalent au nombre d'individu évalué à chaque itération avec la version hypothétique de GAAM), il n'apporte donc pas de gain de temps. Nous avons testé (avec les mêmes paramètres que précédemment) mgaam avec l'algorithme 14 sur le jeu de données micro-mass, les résultats sont présentés dans les tableaux 4.12. Il y a une nette amélioration des résultats.

Algorithm 14 : **Calc_tm_imp** mgaam_pop_size : taille de la population, gaam_pop_size taille de la population de comparaison, T la taille du chromosome, G le nombre de générations

INPUT : *mgaam_pop_size :int, gaam_pop_size :int, T :int,G :int*

tm $\leftarrow \frac{gaam\_pop\_size - mgaam\_pop\_size}{G*T*mgaam\_pop\_size} + \frac{gaam\_pop\_size}{mgaam\_pop\_size}$

OUTPUT *tm*

TABLE 4.12 – Résultat sur Micro-mass en utilisant l'algorithme 14 pour calculer le taux de mutation

Jeux de données/Taille Population		10	20	30	40	60
Micro-mass	Moy	88.588	88.861	**89.216**	89.155	89.155
	Me	88.472	88.888	**89.444**	89.027	89.444
	Max	88.588	91.944	**92.222**	91.666	**92.222**

Enfin, dans une dernière expérience, nous avons réutilisé la méthodologie d'évaluation non biaisée décrite dans la section 4.1 pour comparer GAAM et mGAAM (utilisant 60 individus initiaux et l'algorithme 14 pour le calcul du taux de mutation). Les résultats sont présentés dans les tableaux 4.17 et 4.18. On y constate que les résultats de mGAAM sont similaire à ceux de GAAM sur 11 jeux de données, mais significativement meilleurs sur 6, ce qui confirme l'efficacité de la méthode proposée.

4.4.5 Test sur les vecteurs de meta-données

Jusqu'à présent, tous nos tests ont toujours été effectués sur des exemples fictifs pour démontrer l'efficacité de la méthode. Nous allons dans cette section tester mGAAM sur les vecteurs de taille 512 que nous avons utilisés pour la prédiction du type sémantique dans la section 3.3. Afin de rendre le temps de calcul raisonnable nous n'allons pas utiliser Catboost comme classifieur mais une forêt d'arbres décisionnels de type Extratree.

TABLE 4.13 – Résultat de l'algorithme mGAAM sur plusieurs jeux de données comparativement à GAAM utilisant 10 individus, 10 caractéristiques et 100 itérations, les résultats sont obtenus sur 50 simulations part a1.

Jeux de données/Taille Pop		2	4	6	8	10
Leaf	Moy	73.227	73.220	73.250	73.392	73.333
	Me	73.552	73.552	73.552	73.552	73.552
	Max	73.552	73.552	73.552	73.552	73.552
Thoracic-surgery	Moy	79.590	84.246	83.233	84.766	84.953
	Me	84.254	84.257	85.106	85.106	85.106
	Max	**85.319**	**85.319**	**85.319**	**85.319**	**85.319**
Credit-g	Moy	75.053	75.383	75.291	75.399	75.461
	Me	75.250	**75.599**	**75.599**	**75.599**	**75.599**
	Max	**75.899**	**75.899**	**75.899**	**75.899**	**75.899**
Climate-model	Moy	92.107	92.107	92.344	92.388	92.437
	Me	91.851	92.037	92.407	92.407	92.592
	Max	**92.962**	**92.962**	**92.962**	**92.962**	**92.962**
Dermatology	Moy	96.155	96.704	96.899	96.866	96.844
	Me	96.659	96.935	96.935	96.935	96.935
	Max	**98.046**	**98.046**	**98.046**	**98.046**	**98.046**
Ionosphere	Moy	87.054	88.501	88.632	88.689	88.552
	Me	88.319	88.603	88.888	88.888	88.603
	Max	**89.458**	**89.458**	**89.458**	**89.458**	**89.458**
Audit	Moy	87.722	88.307	87.951	88.858	89.664
	Me	89.432	89.559	89.496	89.559	89.689
	Max	90.142	90.142	90.142	90.142	90.142
SPECTF	Moy	81.810	81.896	81.942	82.164	82.131
	Me	81.951	81.951	81.951	82.234	82.228
	Max	**83.092**	**83.092**	**83.092**	**83.092**	**83.092**

TABLE 4.14 – Résultat de l'algorithme mGAAM sur plusieurs jeux de données comparativement à GAAM utilisant 10 individus, 10 caractéristiques et 100 iterations, les résultats sont obtenus sur 50 simulations part a2.

Jeux de données/Taille Pop		20	30	40	60
Leaf	Moy	73.534	73.504	73.510	**73.552**
	Me	73.552	73.552	73.552	73.552
	Max	73.552	73.552	73.552	73.552
Thoracic-surgery	Moy	85.119	85.217	85.238	**85.285**
	Me	**85.319**	**85.319**	**85.319**	**85.319**
	Max	**85.319**	**85.319**	**85.319**	**85.319**
Credit-g	Moy	75.505	75.563	75.555	**75.623**
	Me	**75.599**	**75.599**	**75.599**	**75.599**
	Max	**75.899**	**75.899**	**75.899**	**75.899**
Climate-model	Moy	92.640	92.725	92.748	**92.862**
	Me	92.777	**92.962**	**92.962**	**92.962**
	Max	**92.962**	**92.962**	**92.962**	**92.962**
Dermatology	Moy	96.990	97.118	97.083	**97.207**
	Me	96.935	96.935	96.935	**97.207**
	Max	**98.046**	**98.046**	**98.046**	**98.046**
Ionosphere	Moy	88.957	88.991	89.065	**89.105**
	Me	88.888	88.888	**89.173**	**89.173**
	Max	**89.458**	**89.458**	**89.458**	**89.458**
Audit	Moy	89.745	89.784	89.802	**89.808**
	Me	89.818	**89.883**	**89.883**	89.819
	Max	**90.142**	90.142	**90.142**	90.142
SPECTF	Moy	82.406	82.463	82.423	**82.531**
	Me	82.517	**82.520**	**82.520**	**82.520**
	Max	**83.092**	**83.092**	**83.092**	**83.092**

TABLE 4.15 – Résultat de l'algorithme mGAAM sur plusieurs jeux de données comparativement à GAAM utilisant 10 individus, 10 caractéristiques et 100 iterations, les résultats sont obtenus sur 50 simulations part b1.

Jeux de données/Taille Pop		2	4	6	8	10
Hill-valley	Moy	52.306	52.369	52.391	52.422	52.420
	Me	52.310	52.392	52.392	52.392	52.392
	Max	**52.640**	**52.640**	**52.640**	**52.640**	**52.640**
Spectrometer	Moy	57.212	57.370	57.578	57.401	57.589
	Me	57.250	57.438	57.532	57.438	57.721
	Max	58.568	**58.945**	**58.945**	**58.945**	**58.945**
Musk	Moy	95.060	95.233	95.124	95.267	95.294
	Me	95.141	95.217	95.240	95.285	95.308
	Max	**95.452**	**95.452**	**95.452**	**95.452**	**95.452**
Semeion	Moy	66.367	66.178	66.204	**66.817**	66.726
	Me	66.823	66.478	66.603	66.917	**66.980**
	Max	**68.926**	68.424	68.361	68.738	68.424
Madelon	Moy	63.856	63.833	63.895	64.039	63.963
	Me	63.653	63.730	63.692	63.884	63.961
	Max	65.038	65.577	65.347	**65.808**	65.731
Har	mean	90.949	**90.966**	90.957	90.842	90.937
	Me	90.994	90.989	**91.018**	90.965	91.013
	Max	91.270	91.261	**91.280**	91.270	91.261
Isolet	mean	74.846	74.979	74.971	74.805	**75.089**
	Me	74.746	75.073	75.099	74.996	**75.022**
	Max	76.221	76.195	76.016	75.990	**76.208**
Parkinson-speech	mean	85.169	85.185	85.433	85.489	85.544
	Me	85.251	85.052	85.582	85.317	85.449
	Max	87.301	87.566	87.169	86.904	87.566
Micro-mass	mean	**89.111**	89.183	88.588	88.649	88.588
	Me	89.166	**89.444**	88.611	88.888	88.472
	Max	91.944	**92.222**	91.944	91.944	91.111

TABLE 4.16 – Résultat de l'algorithme mGAAM sur plusieurs jeux de données comparativement à GAAM utilisant 10 individus, 10 caractéristiques et 100 iterations, les résultats sont obtenus sur 50 simulations part b2.

Jeux de données/Taille Pop		20	30	40	60
Hill-valley	Moy	52.447	52.516	52.514	**52.534**
	Me	52.433	**52.557**	**52.557**	**52.557**
	Max	**52.640**	**52.640**	**52.640**	**52.640**
Spectrometer	Moy	57.883	57.902	**58.131**	58.037
	Me	58.003	58.003	**58.192**	58.003
	Max	58.568	**58.945**	**58.945**	**58.945**
Musk	Moy	95.251	95.290	**95.338**	95.284
	Me	95.240	95.346	**95.369**	95.270
	Max	**95.452**	**95.452**	**95.452**	**95.452**
Semeion	Moy	66.514	66.661	66.572	65.546
	Me	66.698	66.917	66.792	65.599
	Max	**68.926**	68.424	68.361	68.047
Madelon	Moy	**64.353**	64.131	63.941	64.078
	Me	**64.212**	63.961	63.827	63.923
	Max	65.615	65.577	65.308	65.346
Har	Moy	90.853	90.761	90.742	90.411
	Me	90.940	90.756	90.804	90.435
	Max	91.261	91.212	91.203	91.096
Isolet	Moy	74.722	74.339	74.296	73.241
	Me	74.868	74.586	74.284	73.252
	Max	75.798	75.926	75.824	75.221
Parkinson-speech	Moy	85.642	85.820	**85.899**	85.396
	Me	85.582	85.780	**85.978**	85.317
	Max	87.037	87.301	**87.962**	87.169
Micro-mass	Moy	88.550	88.838	88.544	87.755
	Me	88.611	88.611	88.611	87.777
	Max	91.666	91.666	91.388	90.833

TABLE 4.17 – Taux de bonne reconnaissance moyen pour GAAM et mGAAM (60 inds, formule 14), 10 caractéristiques, obtenu en répétant 50 fois une séparation 75% apprentissage 25% test part part a.

Données	Leaf	Thora	Credit	Clima	Derma	Ionos	Audit	SPEC
mGAAM	71.1	**84.6**	72.1	**90.7**	**94**	**90.6**	**93.4**	**76.6**
GAAM	**71.4**	84.2	**72.2**	90.5	93.8	90.2	93	76.1

TABLE 4.18 – Taux de bonne reconnaissance moyen pour GAAM et mGAAM (60 inds, formule 14, 10 caractéristiques, obtenu en répétant 50 fois une séparation 75% apprentissage 25% test part b.

Données	Hill	Spect	Musk	Seme	Made	Har	Isol	Parki	Micro
mGAAM	50	50	99.5	**63.4**	**60.6**	91	73.9	**82.1**	**86.1**
GAAM	50	**50.2**	99.5	63	60.5	91	73.9	80.9	86

Le classifieur utilise 50 estimateurs, une profondeur maximale de 9, un minimum d'individus pour effectuer une séparation de 10, le nombre de caractéristiques explorées à chaque itérations est $\sqrt{512}$, de plus les arbres sont entraînés sur des échantillons (tirage avec remise) représentant 20% de la taille du jeu de données qui leur est présenté. Enfin nous n'utilisons pas pour l'entraînement la totalité des données mais 20% de celles-ci.

Nous avons ensuite utilisé l'algorithme mGAAM avec 60 individus et 70 générations dans le but de conserver 20 caractéristiques. Une fois les 20 caractéristiques identifiées nous avons réentrainé le classifieur avec celles-ci mais en utilisant cette fois tous le jeu de données. Pour comparer nous avons fait la même chose en utilisant toutes les caractéristiques.

Le modèle de base affiche 81% de taux de bonne reconnaissance sur les données de test. Le modèle n'utilisant quant à lui que 20 caractéristiques présente un taux de bonne reconnaissance de 85%. Ces résultats restent très inférieurs à ceux fournis par Catboost mais démontrent l'intérêt de la méthode pour améliorer le taux de bonne reconnaissance tout en diminuant le temps de calcul dédié au calcul des caractéristiques.

4.4.6 Conclusion

Dans cette section, nous avons présenté une modification de l'algorithme GAAM. La principale modification consiste à diminuer le nombre de mutations que subit chaque individu tout en utilisant une plus grande population.

Il ressort des expériences que nous avons réalisées que notre algorithme présente deux avantages par rapport à l'algorithme original. Premièrement, une amélioration du taux de bonne reconnaissance. Deuxièmement, l'utilisation d'une plus grande population réduit le temps de calcul, en diminuant le nombre total d'individus à réévaluer à chaque itération. Dans le cas de jeux de données contenant un grand nombre de caractéristiques, nous avons introduit une deuxième possibilité pour calculer le taux de mutation. Celle-ci n'a pas l'avantage d'un gain de temps mais permet de maintenir de meilleurs résultats que l'algorithme original à coût de calcul égal. De futures études devraient être menées pour combiner notre méthode avec les techniques implémentées dans fGAAM [RJ21] pour accélérer l'algorithme. De plus, nous pourrions être intéressés à combiner notre nouvelle méthode avec les techniques d'ensemencement de la population initiale [CRB+21] présentées dans la section précédente afin de voir si les deux méthodes se combinent pour donner de meilleurs résultats.

4.5 Conclusion du chapitre

Dans ce chapitre nous avons présenté le problème de la sélection de caractéristiques pour l'apprentissage automatique. Ce problème est intervenu dans le cadre de nos recherches sur le profilage des données quand nous nous sommes retrouvés confrontés à des vecteurs de caractéristiques de grandes dimensions. Ainsi, partant de l'état de l'art et de notre problème, nous avons choisi d'explorer des solutions basées sur les algorithmes génétiques. Les recherches que nous avons menées dans ce cadre visaient à améliorer les performances d'algorithmes existants dans le désir de les appliquer ensuite à nos problématiques de profilage des données.

Cependant, des recherches doivent être encore faites afin de rendre le procédé plus rapide. En effet, les modèles d'apprentissage automatique qui donnent les meilleurs résultats sont relativement lents à entraîner ce qui limite dans ce cas l'usage des algorithmes génétiques qui demandent d'effectuer le processus d'apprentissage un grand nombre de fois. Bien que la tâche soit aisément parallélisable, le coût reste important. Afin de rendre le processus plus facilement applicable des études doivent être menées sur les deux points suivants : la possibilité d'utiliser un modèle de substitution et la possibilité d'écarter des dimensions du processus de sélection afin de diminuer la taille l'espace de recherche.

CHAPITRE 5
CONCLUSION

5.1 Contributions

Dans cette thèse nous avons présenté le domaine du profilage des données tabulaires. Ce domaine est principalement subdivisé en deux sous-domaines : le profilage monocolonne d'une part, et le profilage multi-colonnes d'autre part. Nous nous sommes intéressés au profilage monocolonne. Ce champ d'étude consiste à extraire des métadonnées des données afin de les résumer ou de les décrire. Nous avons utilisé un grand nombre caractéristiques simples issues du profilage, représentées sous forme de vecteurs afin de réaliser des tâches complexes.
La première activité de profilage complexe que nous avons abordé est la découverte du type sémantique de données tabulaires. Nous avons choisi une approche utilisant l'apprentissage automatique pour traiter cette problématique. En effet, ce type de méthode est considéré comme plus robuste aux bruits dans les données que les techniques traditionnellement utilisées. Notre contribution vise à pallier plusieurs soucis des méthodes utilisant l'apprentissage artificiel pour effectuer cette tâche.

Tout d'abord, les méthodes de l'état de l'art utilisent un très grand volume de données réelles pour réaliser le processus d'apprentissage. Or, il est rarement possible d'obtenir les quantités de données requises de manière simple. Nous avons donc proposé une méthode de génération de données synthétiques permettant de réduire le nombre de données réelles à utiliser, rendant de ce fait la méthode plus accessible à un usage industriel. De plus l'utilisation de données synthétiques a pour effet de rendre la méthode facilement adaptable à de nouveaux types sémantiques où peu de données réelles sont disponibles. Il n'y a pas besoin d'avoir des milliers de colonnes d'exemples réels, quelques colonnes représentant la plupart des valeurs possiblement prises par le domaine sémantique sont suffisantes pour générer un grand nombre de colonnes synthétiques.

De plus, nous avons modifié les caractéristiques utilisées traditionnelle-

ment dans l'état de l'art afin qu'elles ne nécessitent plus l'usage d'une modèle de plongement de mots. En effet, les modèles de ce type sont intrinsèquement cantonnés à une langue ; ce sont généralement des modèles coûteux à entraîner et non disponibles dans toutes les langues.

Constatant que ces caractéristiques utilisées pour la détection des types sémantiques étaient puissantes, nous avons décidé d'étendre leur usage à d'autres problématiques du monde de la donnée. Tout d'abord, celle de la détection de colonnes de données quasi dupliquées. Dans la littérature un grand intérêt est porté à l'identification d'enregistrements dupliqués et à l'identification d'une même entité dans plusieurs jeux de données. Cependant peu de travaux ont été effectués sur la détection de colonnes quasi dupliquées, nous avons dans cette thèse proposé une méthode de génération d'exemples et une méthode d'apprentissage pour identifier deux colonnes quasi dupliquées. Notre méthode ne nécessite pas de fixer un seuil explicite, le seuil étant appris à partir des exemples générés, en modifiant la méthode de génération on peut ainsi faire varier la définition d'une colonne quasi dupliquée. Nous avons ensuite étendu ce concept aux jeux de données quasi dupliquées en utilisant le même principe. En effet en concaténant l'ensemble des colonnes d'un jeu de données on peut se ramener à un cas proche de celui d'une seul colonne.

Lors de nos études sur le profilage des données, nous nous sommes confrontés au problème de sélection des caractéristiques. Le but étant de sélectionner un sous-ensemble de caractéristiques (en précisant la taille de celui-ci ou non) qui va fournir les meilleurs résultats possibles lors des prédictions (selon la métrique d'évaluation choisie). Nous nous sommes intéressés à ce problème car pour des raisons d'optimisation (afin d'éviter le problème de la malédiction des grandes dimensions) et de temps de calcul (chaque caractéristique coûte du temps à calculer), il est préférable de sélectionner un sous-ensemble des caractéristiques initiales avant d'effectuer l'apprentissage final. Sachant que dans notre cas de nombreuses caractéristiques n'ont que peu d'utilité mais il est difficile de savoir à l'avance celles qui vont être importantes ou non.

Dans le cadre de cette thèse nous avons choisi de traiter cette problématique en utilisant des algorithmes génétiques. Les algorithmes génétiques sont une catégorie de métaheuristique à base de population, ils permettent d'obtenir des solutions approximatives à des problèmes d'optimisations complexes. Nous avons choisi d'utiliser les AG car ils possèdent l'avantage d'être adaptables à tous les classifieurs, sont très performant et offrent des résultats facilement explicables.

Nous avons proposé plusieurs innovations par rapport à l'état de l'art. Tout d'abord, nous avons développé une méthode d'initialisation non aléa-

toire de la population de départ des algorithmes génétiques. En effet, la population initiale des algorithmes génétiques est traditionnellement créée de manière aléatoire. Cependant, il s'est avéré que l'injection d'individus issus d'un premier processus de sélection avait un impact positif sur les résultat finaux dans d'autres domaines que celui de la sélection de caractéristiques [MRSB13].

Nous avons ainsi créé un procédé de génération d'individus supposés plus optimaux que s'ils étaient sélectionnés aléatoirement. Nous avons ensuite montré qu'injecter une part de ce type d'individus non aléatoire dans la population initiale permettait dans certains cas d'améliorer les résultats globaux des algorithmes génétiques pour la sélection de caractéristiques ; et ce principalement quand l'espace de recherche est très grand.

Nous nous sommes ensuite intéressés à l'algorithme GAAM (algorithme génétique avec des mutations agressives) qui est un algorithme génétique spécialement conçu pour la sélection de caractéristiques. Celui-ci a pour particularité d'utiliser un encodage entier et un faible nombre d'individus qui sont peu croisés mais énormément mutés.

Nous avons montré que la version de l'algorithme proposé pouvait être améliorée avec quelques changements tout en gardant un coût en calcul égal. Pour ce faire nous avons proposé d'agrandir la population initiale de l'algorithme et de contrebalancer cette augmentation par une diminution du taux de mutation. Il en résulte globalement de meilleurs résultats que la version classique de GAAM.

Ces résultats nous ont permis de confirmer notre choix d'utiliser un algorithme génétique pour traiter cette problématique mais d'autres études sont nécessaires pour rendre le processus applicable sur les vecteurs que nous utilisons dans le cadre du profilage.

5.2 Futures directions

Dans le cadre de l'identification automatique des types sémantiques nous envisageons plusieurs possibilités d'amélioration. Tout d'abord afin d'améliorer les résultats de notre algorithme de détection nous considérons qu'il est impératif de construire un système hybride utilisant de l'apprentissage artificiel mais aussi des expressions régulières et des dictionnaires. En effet, quand le nombre de valeurs distinctes dans une colonne est faible par rapport à l'ensemble des valeurs réellement possibles, les algorithmes d'apprentissage automatique ont de faibles performances. Par exemple, si l'on s'intéresse à une colonne de noms de pays et que celle-ci ne contient que 2 noms de pays, il va être difficile pour notre méthode d'identifier la colonne comme étant

de type sémantique "nom de pays", alors que pour une méthode utilisant des dictionnaires ce cas est simple. Ainsi nous nous intéresserons à développer un système hybride utilisant les deux méthodes, celui-ci privilégiera en fonction du cas à traiter la meilleure méthode à utiliser. De plus dans un soucis de fiabilisation des résultats il serait intéressant d'utiliser les résultats combinés des deux méthodes pour créer les prédictions finales. Par exemple commencer par effectuer une prédiction avec une méthode utilisant de l'apprentissage artificiel et la confirmer sur un échantillon de la colonne à l'aide d'expressions régulières (ce qui devient plus facile puisque l'on suspecte déjà quel type sémantique rechercher). Deux autres pistes de recherche dans le domaine que nous souhaitons aussi explorer sont la découverte découverte automatique de nouvelles catégories (à l'aide d'informations extérieurs) ainsi que l'identification de type sémantique multiple au sein d'une colonne (colonnes hétérogènes).

Pour toutes les applications utilisant les vecteurs de caractéristiques que nous avons présentés on s'intéressera à ajouter de nouvelles caractéristiques. Nous chercherons ainsi tout d'abord à créer de nouvelles caractéristiques plus centrées sur les valeurs numériques. Ensuite nous souhaitons explorer la possibilité d'extraire des caractéristiques à partir de comptage sur des duos de lettres et réduire la dimension du vecteur obtenu à l'aide d'un auto-encodeur. Enfin on cherchera à utiliser des techniques d'apprentissage des représentations [ZLLZ20] pour apprendre des caractéristiques directement depuis les données.

Pour la sélection de caractéristiques nous avons pour ambition d'améliorer notre méthode d'ensemencement de la population initiale en de multipliant les sources d'individus. Par exemple, on pourra ajouter des caractéristiques issues de la méthode Lasso [Tib96] et de la méthode SFS [AB96]. De plus, nous mènerons des recherches afin d'évaluer les effets de ces nouvelles méthodes sur notre version améliorée de l'algorithme GAAM. La population initiale étant plus grande avec notre méthode on peut s'attendre à des résultats différents à l'aide de l'initialisation plutôt qu'avec l'algorithme GAAM classique. Nous avons aussi pour objectif d'améliorer notre version de l'algorithme GAAM. Tout d'abord en proposant une méthode pour sélectionner automatiquement la taille de la population initiale. Ensuite on s'intéressera à modifier l'algorithme pour éliminer les caractéristiques les moins pertinentes au fur et à mesure des générations afin de progressivement diminuer l'espace de recherche. Nous chercherons aussi à modifier la phase de sélection des individus d'une génération à l'autre pour favoriser une plus grande diversité dans la population. Enfin pour limiter le temps de calcul en cas d'usage d'un classifieur coûteux à entraîner, on s'intéressera aux possibilités d'utilisation d'un modèle de substitution [Mol20] pour accélérer une partie des calculs.

REFERENCES

[AAGM21] Prachi Agrawal, Hattan F. Abutarboush, Talari Ganesh, and Ali Wagdy Mohamed. Metaheuristic algorithms on feature selection : A survey of one decade of research (2009-2019). *IEEE Access*, 9 :26766–26791, 2021.

[AB96] David W. Aha and Richard L. Bankert. A comparative evaluation of sequential feature selection algorithms. In *Learning from Data*, pages 199–206. Springer New York, 1996.

[ABE+55] Miriam Ayer, H. D. Brunk, G. M. Ewing, W. T. Reid, and Edward Silverman. An Empirical Distribution Function for Sampling with Incomplete Information. *The Annals of Mathematical Statistics*, 26(4) :641 – 647, 1955.

[AFH22] Shatha Awawdeh, Hossam Faris, and Hazem Hiary. Evoimputer : An evolutionary approach for missing data imputation and feature selection in the context of supervised learning. *Knowledge-Based Systems*, 236 :107734, 2022.

[Agg16] C.C. Aggarwal. *Outlier Analysis*, pages 219–247. Springer International Publishing, 2016.

[AGN15] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. Profiling relational data : a survey. *The VLDB Journal*, 24(4) :557–581, June 2015.

[AGNP18] Ziawasch Abedjan, Lukasz Golab, Felix Naumann, and Thorsten Papenbrock. Data profiling. *Synthesis Lectures on Data Management*, 10(4) :1–154, November 2018.

[AGO+13] D. Anguita, Alessandro Ghio, L. Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *ESANN*, 2013.

[AI08] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1) :117–122, January 2008.

[AMS99] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1) :137–147, 1999.

[AP21] Sercan Ö. Arik and Tomas Pfister. Tabnet : Attentive interpretable tabular learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(8) :6679–6687, May 2021.

[BB15] Abhijit Bendale and Terrance Boult. Towards open world recognition. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2015.

[BE19] Laure Berti-Equille. Reinforcement learning for data preparation with active reward learning. In Samira El Yacoubi, Franco Bagnoli, and Giovanna Pacini, editors, *Internet Science*, pages 121–132, Cham, 2019. Springer International Publishing.

[Ben38] Frank Benford. The law of anomalous numbers. 78(4), 1938.

[BFI+22] Lukas Budach, Moritz Feuerpfeil, Nina Ihde, Andrea Nathansen, Nele Noack, Hendrik Patzlaff, Hazar Harmouch, and Felix Naumann. The effects of data quality on machine learning performance, 2022. working paper or preprint https ://arxiv.org/abs/2207.14529.

[BFOS17] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification And Regression Trees*. Routledge, oct 2017.

[BHK20] Avrim Blum, John Hopcroft, and Ravindran Kannan. *Foundations of Data Science*, pages 12–32. Cambridge University Press, 2020.

[Bis19] Ekaba Bisong. *Building Machine Learning and Deep Learning Models on Google Cloud Platform : A Comprehensive Guide for Beginners*, pages 59–64. Apress, Berkeley, CA, 2019.

[BJB19] Abdelkader Berrouachedi, Rakia Jaziri, and Gilles Bernard. Deep extremely randomized trees. In Tom Gedeon, Kok Wai Wong, and Minho Lee, editors, *Neural Information Processing*, pages 717–729, Cham, 2019. Springer International Publishing.

[BKN17] Tobias Bleifuss, Sebastian Kruse, and Felix Naumann. Efficient denial constraint discovery with hydra. *Proc. VLDB Endow.*, 11(3) :311–323, nov 2017.

[Bre01] Leo Breiman. Random forests. *Machine Learning*, 45(1) :5–32, Oct 2001.

[Bro80] Michael L. Brodie. Data quality in information systems. *Information & Management*, 3(6) :245–258, 1980.

[Bro00] Andrei Z. Broder. Identifying and filtering near-duplicate documents. In Raffaele Giancarlo and David Sankoff, editors, *Combinatorial Pattern Matching*, pages 1–10, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[Bur19] A. Burkov. *The Hundred-Page Machine Learning Book*, pages 2–7. Andriy Burkov, 2019.

[CBG+21] Marc Chevallier, Faouzi Boufares, Nistor Grozavu, Nicoleta Rogovschi, and Charly Clairmont. Near duplicate column identification : a machine learning approach. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, December 2021.

[CDP16] Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. Relaxed functional dependencies—a survey of approaches. *IEEE Transactions on Knowledge and Data Engineering*, 28(1) :147–165, 2016.

[CGB+22] Marc Chevallier, Nistor Grozavu, Faouzi Boufarès, Nicoleta Rogovschi, and Charly Clairmont. Trade between population size and mutation rate for gaam (genetic algorithm with aggressive mutation) for feature selection. In Ilias Maglogiannis, Lazaros Iliadis, John Macintyre, and Paulo Cortez, editors, *Artificial Intelligence Applications and Innovations*, pages 432–444, Cham, 2022. Springer International Publishing.

[CIP13] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. Discovering denial constraints. *Proc. VLDB Endow.*, 6(13) :1498–1509, aug 2013.

[CKMS06] Graham Cormode, Flip Korn, S. Muthukrishnan, and Divesh Srivastava. Space- and time-efficient deterministic algorithms for biased quantiles over data streams. In *Proceedings of the Twenty-Fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '06, page 263–272, New York, NY, USA, 2006. Association for Computing Machinery.

[CP09] Peter Christen and Agus Pudjijono. Accurate synthetic generation of realistic personal information. In *Advances in Knowledge Discovery and Data Mining*, pages 507–514. Springer Berlin Heidelberg, 2009.

[CRB+21] Marc Chevallier, Nicoleta Rogovschi, Faouzi Boufarès, Nistor Grozavu, and Charly Clairmont. Seeding initial population, in genetic algorithm for features selection. In Ajith Abraham, Yukio Ohsawa, Niketa Gandhi, M.A. Jabbar, Abdelkrim Haqiq, Seán McLoone, and Biju Issac, editors, *Proceedings of the 12th International Conference on Soft Computing and Pattern Recognition (SoCPaR 2020)*, pages 572–582, Cham, 2021. Springer International Publishing.

[CRB+22a] Marc Chevallier, Nicoleta Rogovschi, Faouzi Boufarès, Nistor Grozavu, and Charly Clairmont. Detecting near duplicate dataset. In *Proceedings of the 13th International Conference on Soft Computing and Pattern Recognition (SoCPaR 2021)*, pages 394–403. Springer International Publishing, 2022.

[CRB+22b] Marc Chevallier, Nicoleta Rogovschi, Faouzi Boufarès, Nistor Grozavu, and Charly Clairmont. Detecting near duplicate dataset with machine learning. *International Journal of Computer Information Systems and Industrial Management Applications*, 14(32) :374–385, 07 2022.

[CT10] Gavin C. Cawley and Nicola L.C. Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *J. Mach. Learn. Res.*, 11 :2079–2107, aug 2010.

[CV13] Peter Christen and Dinusha Vatsalan. Flexible and extensible generation and corruption of personal data. In *Proceedings of the 22nd ACM international conference on Conference on information &amp knowledge management - CIKM '13*. ACM Press, 2013.

[DG17] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[DGB18] Akshay Raj Dhamija, Manuel Günther, and Terrance E. Boult. Reducing network agnostophobia. In *NeurIPS*, pages 9175–9186, 2018.

[DHBS22] Till Döhmen, Madelon Hulsebos, Christian Beecks, and Sebastian Schelter. Gitschemas : A dataset for automating relational

data preparation tasks. In *2022 IEEE 38th International Conference on Data Engineering Workshops (ICDEW)*, pages 74–78, 2022.

[DJL+13] Dong Deng, Yu Jiang, Guoliang Li, Jian Li, and Cong Yu. Scalable column concept determination for web tables using large knowledge bases. *Proc. VLDB Endow.*, 6(13) :1606–1617, aug 2013.

[EZH16] E. Emary, Hossam M. Zawbaa, and Aboul Ella Hassanien. Binary grey wolf optimization approaches for feature selection. *Neurocomputing*, 172 :371–381, January 2016.

[Fan08] Wenfei Fan. Dependencies revisited for improving data quality. In *Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems - PODS '08*, New York, New York, USA, 2008. ACM Press.

[FGJK08] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *ACM Transactions on Database Systems*, 33(2) :1–48, June 2008.

[F.R01] Karl Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11) :559–572, 1901.

[FS99] Yoav Freund and Robert E. Schapire. A short introduction to boosting. In *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1401–1406. Morgan Kaufmann, 1999.

[FvRK+19] Matthias Feurer, Jan N. van Rijn, Arlind Kadra, Pieter Gijsbers, Neeratyoy Mallik, Sahithya Ravi, Andreas Müller, Joaquin Vanschoren, and Frank Hutter. Openml-python : an extensible python api for openml. 2019.

[GBV20] Margherita Grandini, Enrico Bagli, and Giorgio Visani. Metrics for multi-class classification : an overview. working paper or preprint https ://arxiv.org/abs/2008.05756, 2020.

[GGHD04] Isabelle Guyon, Steve Gunn, Asa Ben Hur, and Gideon Dror. Result analysis of the nips 2003 feature selection challenge. In *Proceedings of the 17th International Conference on Neural Information Processing Systems*, NIPS'04, page 545–552, Cambridge, MA, USA, 2004. MIT Press.

[GK20] Sainyam Galhotra and Udayan Khurana. Semantic search over structured data. In *Proceedings of the 29th ACM International Conference on Information Knowledge Management*, CIKM '20, page 3381–3384, New York, NY, USA, 2020. Association for Computing Machinery.

[GKH+19] Sainyam Galhotra, Udayan Khurana, Oktie Hassanzadeh, Kavitha Srinivas, Horst Samulowitz, and Miao Qi. Automated feature enhancement for predictive modeling using external knowledge. In *2019 International Conference on Data Mining Workshops (ICDMW)*, pages 1094–1097, 2019.

[GOV22] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on tabular data ? working paper or preprint https ://hal.archives-ouvertes.fr/hal-03723551, July 2022.

[Gro09] Nistor Grozavu. *Classification Topologique pondérée : approches modulaires, hybrides et collaboratives*. PhD thesis, Université Paris 13, 01 2009.

[Grö19] U Grömping. South german credit data : Correcting a widely used data set. *Reports in Mathematics*, 2019.

[GWBV02] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1) :389–422, 01 2002.

[HAA+19] Ahmad Hassanat, Khalid Almohammadi, Esra'a Alkafaween, Eman Abunawas, Awni Hammouri, and V. B. Surya Prasath. Choosing mutation and crossover ratios for genetic algorithms—a review with a new dynamic approach. *Information*, 10(12), 2019.

[Hal00] Mark A. Hall. Correlation-based feature selection of discrete and numeric class machine learning. Technical report, 2000. Working Paper.

[Har20] Hazar Harmouch. *Single-column data profiling*. doctoral thesis, Universität Potsdam, 2020.

[HBR18] Nishtha Hooda, Seema Bawa, and Prashant Singh Rana. Fraudulent firm classification : A case study of an external audit. *Applied Artificial Intelligence*, 32(1) :48–64, January 2018.

[HGG+22] Madelon Hulsebos, Sneha Gathani, James Gale, Isil Dillig, Paul T. Groth, and Caugatay Demiralp. Making table understanding work in practice. *ArXiv*, abs/2109.05173, 2022.

[HGH+19] Kevin Hu, Snehalkumar 'Neil' S. Gaikwad, Madelon Hulsebos, Michiel A. Bakker, Emanuel Zgraggen, César Hidalgo, Tim Kraska, Guoliang Li, Arvind Satyanarayan, and Çağatay Demiralp. VizNet. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, May 2019.

[HHB+19] Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Cagatay Demiralp, and César Hidalgo. Sherlock. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, July 2019.

[HKPT99] Yká Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. Tane : An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(2) :100–111, 1999.

[HKW01] F. Hussein, N. Kharma, and R. Ward. Genetic algorithms for feature selection and weighting, a review and study. In *Proceedings of Sixth International Conference on Document Analysis and Recognition*. IEEE Comput. Soc, 2001.

[HMAB+20] Abdelaziz I. Hammouri, Majdi Mafarja, Mohammed Azmi Al-Betar, Mohammed A. Awadallah, and Iyad Abu-Doush. An improved dragonfly algorithm for feature selection. *Knowledge-Based Systems*, 203 :106131, September 2020.

[HNSS95] Peter J. Haas, Jeffrey F. Naughton, S. Seshadri, and Lynne Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *Proceedings of the 21th International Conference on Very Large Data Bases*, VLDB '95, page 311–322, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[HR03] Geoffrey E Hinton and Sam Roweis. Stochastic neighbor embedding. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15. MIT Press, 2003.

[IC19] Ihab F. Ilyas and Xu Chu. *Data Cleaning*. Association for Computing Machinery, New York, NY, USA, 2019.

[IdTFM18] Andrew Ilyas, Joana M. F. da Trindade, Raul Castro Fernandez, and Samuel Madden. Extracting syntactical patterns from databases. *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 41–52, 2018.

[Jac12] Paul Jaccard. The distribution of the flora in the alpine zone.1. *New Phytologist*, 11(2) :37–50, 1912.

[Jar89] Matthew A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406) :414–420, June 1989.

[JH10] Xin Jin and Jiawei Han. *K-Means Clustering*, pages 563–564. Springer US, Boston, MA, 2010.

[JLLC13] Zhongming Jin, Cheng Li, Yue Lin, and Deng Cai. Density sensitive hashing. *IEEE transactions on cybernetics*, 44, 10 2013.

[KG21] Udayan Khurana and Sainyam Galhotra. Semantic concept annotation for tabular data. In *Proceedings of the 30th ACM International Conference on Information &amp Knowledge Management*. ACM, October 2021.

[KMAB13] Angel Kuri-Morales and Edwin Aldana-Bobadilla. The best genetic algorithm i. In Félix Castro, Alexander Gelbukh, and Miguel González, editors, *Advances in Soft Computing and Its Applications*, pages 1–15, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[KMF+17] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm : A highly efficient gradient boosting decision tree. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[KN03] Jaewoo Kang and Jeffrey F. Naughton. On schema matching with opaque column names and data values. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, page 205–216, New York, NY, USA, 2003. Association for Computing Machinery.

[Knu73] Donald E Knuth. *The art of computer programming : Sorting and searching v. 3*, pages 391–392. Addison-Wesley series in computer science and information processing. Addison Wesley, London, England, January 1973.

[KPHH11] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Wrangler : Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, page

3363–3372, New York, NY, USA, 2011. Association for Computing Machinery.

[LHK04] Mong Li Lee, W. Hsu, and Vijay Kothari. Cleaning the spurious links in data. *IEEE Intelligent Systems*, 19(2) :28–33, 2004.

[LKT+13] D. D. Lucas, R. Klein, J. Tannahill, D. Ivanova, S. Brandon, D. Domyancic, and Y. Zhang. Failure analysis of parameter-induced simulation crashes in climate models. *Geoscientific Model Development*, 6(4) :1157–1171, 2013.

[LM12] Huan Liu and Hiroshi Motoda. *Feature selection for knowledge discovery and data mining*, volume 454, pages 73–90. Springer Science & Business Media, 2012.

[LM14] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1188–1196, Bejing, China, 22–24 Jun 2014. PMLR.

[LRU20] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*, page 77. Cambridge University Press, 3 edition, 2020.

[LSSB21] Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. Towards learned metadata extraction for data lakes. In Kai-Uwe Sattler, Melanie Herschel, and Wolfgang Lehner, editors, *BTW 2021*, pages 325–336. Gesellschaft für Informatik, Bonn, 2021.

[LW66] G. N. Lance and W. T. Williams. Computer Programs for Hierarchical Polythetic Classification (“Similarity Analyses”). *The Computer Journal*, 9(1) :60–64, 05 1966.

[MG82] Jayadev Misra and David Gries. Finding repeated elements. *Sci. Comput. Program.*, 2 :143–152, 1982.

[Mol20] Christoph Molnar. *Interpretable Machine Learning*, pages 136–137. Lulu.com, Morrisville, NC, February 2020.

[MRSB13] Ben Meadows, Patricia Riddle, Cameron Skinner, and Michael M. Barley. Evaluating the seeding genetic algorithm. In Stephen Cranefield and Abhaya Nayak, editors, *AI 2013 : Advances in Artificial Intelligence*, pages 221–227, Cham, 2013. Springer International Publishing.

[Mus17] Suleiman Mustafa. Feature selection using sequential backward method in melanoma recognition. In *2017 13th International Conference on Electronics, Computer and Computation (ICECCO)*, pages 1–4, 2017.

[Nav01] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1) :31–88, mar 2001.

[NMC05] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning - ICML '05*. ACM Press, 2005.

[NTT11] Kai Wang Ng, Guo-Liang Tian, and Man-Lai Tang. *Dirichlet and Related Distributions*, pages 37–95. Wiley, April 2011.

[OCD+14] E. Osaba, R. Carballedo, F. Diaz, E. Onieva, P. Lopez, and A. Perallos. On the influence of using initialization functions on genetic algorithms solving combinatorial optimization problems : A first study on the tsp. In *2014 IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS)*, pages 1–6, 2014.

[Ohl59] Herbert Ohlman. Subject-word letter frequencies with applications to superimposed coding. In *Proceedings of the International Conference on Scientific Information*. National Academies Press, December 1959.

[PEM+15] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. Functional dependency discovery : An experimental evaluation of seven algorithms. *Proc. VLDB Endow.*, 8(10) :1082–1093, jun 2015.

[PGV+18] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost : Unbiased boosting with categorical features. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 6639–6649, Red Hook, NY, USA, 2018. Curran Associates Inc.

[PIP20] George Papadakis, Ekaterini Ioannou, and Themis Palpanas. Entity resolution : Past, present and yet-to-come : From structured to heterogeneous, to crowd-sourced, to deep learned. 2020. EDBT/ICDT 2020 Joint Conference ; Conference date : 20-03-2020 Through 02-04-2020.

[PN16] Thorsten Papenbrock and Felix Naumann. A hybrid approach to functional dependency discovery. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, page 821–833, New York, NY, USA, 2016. Association for Computing Machinery.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe : Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[QB00] Mohamed Quafafou and Moussa Boussouf. Generalized rough sets based feature selection. *Intelligent Data Analysis*, 4 :3–17, 2000. 1.

[QECF+18] Abdulhakim A. Qahtan, Ahmed Elmagarmid, Raul Castro Fernandez, Mourad Ouzzani, and Nan Tang. Fahes : A robust disguised missing values detector. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 2100–2109, New York, NY, USA, 2018. Association for Computing Machinery.

[RCIR17] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. Holoclean : Holistic data repairs with probabilistic inference. *Proc. VLDB Endow.*, 10(11) :1190–1201, August 2017.

[Red98] Thomas C Redman. The impact of poor data quality on the typical enterprise. *Communications of the ACM*, 41(2) :79–82, 1998.

[Red16] Thomas C Redman. Bad data costs the us $3 trillion per year. *Harvard Business Review*, 22 :11–18, 2016.

[Ree10] Colin R Reeves. Genetic algorithms. In *Handbook of metaheuristics*, pages 109–139. Springer, 2010.

[Rej14] Izabela Rejer. Genetic algorithm with aggressive mutation for feature selection in BCI feature space. *Pattern Analysis and Applications*, 18(3) :485–492, nov 2014.

[RJ21] Izabela Rejer and Jarosław Jankowski. fGAAM : A fast and resizable genetic algorithm with aggressive mutation for feature selection. *Pattern Analysis and Applications*, 25(2) :253–269, June 2021.

[RL15] Izabela Rejer and Krzysztof Lorenz. Classic genetic algorithm vs. genetic algorithm with aggressive mutation for feature selection for a brain-computer interface. *Przeglad Elektrotechniczny*, 1 :100–104, 02 2015.

[Roj96] Raúl Rojas. *The Backpropagation Algorithm*, pages 149–182. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[ROvdS11] Thomas Rückstiess, Christian Osendorfer, and Patrick van der Smagt. Sequential feature selection for classification. In Dianhui Wang and Mark Reynolds, editors, *AI 2011 : Advances in Artificial Intelligence*, pages 132–141, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[SdRRSB13] Walter J. Scheirer, Anderson de Rezende Rocha, Archana Sapkota, and Terrance E. Boult. Toward open set recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(7) :1757–1772, 2013.

[Sha05] Y. Shafranovich. Common format and MIME type for comma-separated values (CSV) files. Technical report, October 2005.

[SIS+13] B. E. Sakar, M. E. Isenkul, C. O. Sakar, A. Sertbas, F. Gurgen, S. Delil, H. Apaydin, and O. Kursun. Collection and analysis of a Parkinson speech dataset with multiple types of sound recordings. *IEEE J Biomed Health Inform*, 17(4) :828–834, Jul 2013.

[SLL+22] Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. Annotating columns with pre-trained language models. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 1493–1503, New York, NY, USA, 2022. Association for Computing Machinery.

[SMdS13] Pedro F. B. Silva, André R. S. Marçal, and Rubim M. Almeida da Silva. Evaluation of features for leaf discrimination. In Mohamed Kamel and Aurélio Campilho, editors, *Image Analysis and Recognition*, pages 197–204, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[SR17] Matthew Scherreik and Brian Rigling. Automatic threshold selection for multi-class open set recognition. In Firooz A. Sadjadi and Abhijit Mahalanobis, editors, *SPIE Proceedings*. SPIE, May 2017.

[Sta21] Statista. Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2025, June 2021. The data was taken from various publications released over several years : Forecast for the years 2018 and 2019 as of 2018 ; Forecast for 2020 as of May 2021 ; Forecast for 2021 to 2025 as of March 2021 based on figure for 2020 provided by the source. Figures were rounded to provide a better understanding of the statistic. The figures from 2021 to 2025 were calculated by Statista based on the 2020 forecast figure and the five-year compound annual growth rate (CAGR) of 23 percent provided by the source. The figures prior to 2020 are based on IDC's forecast from late 2018.

[TCH21] Mohamed Trabelsi, Jin Cao, and Jeff Heflin. Selab : Semantic labeling with bert. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021.

[THM+19] Thaer Thaher, Ali Asghar Heidari, Majdi Mafarja, Jin Song Dong, and Seyedali Mirjalili. Binary harris hawks optimizer for high-dimensional, low sample size feature selection. In *Algorithms for Intelligent Systems*, pages 251–272. Springer Singapore, November 2019.

[Tib96] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society : Series B (Methodological)*, 58(1) :267–288, 1996.

[TXZ14] Binh Tran, Bing Xue, and Mengjie Zhang. Overview of particle swarm optimisation for feature selection in classification. In *Lecture Notes in Computer Science*, pages 605–617. Springer International Publishing, 2014.

[VBDB+01] Gitte Vanwinckelen, Hendrik Blockeel, Bernard De Baets, Bernard Manderick, Michaël Rademaker, and Willem Waegeman. On estimating model accuracy with repeated cross-validation, 2012-01-01.

[VC21] CP Vandana and Ajeet A Chikkamannur. Feature selection : An empirical study. *International Journal of Engineering Trends and Technology*, 69(2) :165–170, 2021.

[VCPn13] Max Vladymyrov and Miguel Á. Carreira-Perpiñán. Entropic affinities : Properties and efficient numerical computation. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, page III–477–III–485. JMLR.org, 2013.

[vdMH08] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86) :2579–2605, 2008.

[VvRBT14] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. OpenML. *ACM SIGKDD Explorations Newsletter*, 15(2) :49–60, June 2014.

[VVV98] V.N. Vapnik, V.A. VAPNIK, and V.N. Vapnik. *Statistical Learning Theory*, pages 20–21. A Wiley-Interscience publication. Wiley, 1998.

[WC21] Jacques Wainer and Gavin Cawley. Nested cross-validation when selecting classifiers is overzealous for most practical applications. *Expert Syst. Appl.*, 182(C), nov 2021.

[WGR01] Catharine Wyss, Chris Giannella, and Edward Robertson. Fastfds : A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances extended abstract. In Yahiko Kambayashi, Werner Winiwarter, and Masatoshi Arikawa, editors, *Data Warehousing and Knowledge Discovery*, pages 101–110, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[Win99] W Winkler. The state of record linkage and current research problems, statistics of income division, internal revenue service publication r99/04. 1999.

[Wol92] David H. Wolpert. Stacked generalization. *Neural Networks*, 5(2) :241–259, 1992.

[WSSJ14] Jingdong Wang, Heng Shen, Jingkuan Song, and Jianqiu Ji. Hashing for similarity search : A survey. 08 2014. https ://arxiv.org/abs/1408.2927.

[WYZ16] Yasi Wang, Hongxun Yao, and Sicheng Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184 :232–242, 2016. RoLoD : Robust Local Descriptors for Computer Vision 2014.

[ZE02] Bianca Zadrozny and Charles Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '02*. ACM Press, 2002.

[ZHS+20] Dan Zhang, Madelon Hulsebos, Yoshihiko Suhara, Çağatay Demiralp, Jinfeng Li, and Wang-Chiew Tan. Sato. *Proceedings of the VLDB Endowment*, 13(12) :1835–1848, August 2020.

[ZLLZ20] Haitao Zhao, Zhihui Lai, Henry Leung, and Xianyi Zhang. *Feature learning and understanding : Algorithms and applications*, pages 9–12. 2020.

[ZTLS14] Maciej Zięba, Jakub M. Tomczak, Marek Lubicz, and Jerzy Swiątek. Boosted svm for extracting rules from imbalanced data in application to prediction of the post-operative life expectancy in the lung cancer patients. *Applied Soft Computing*, 14 :99–108, 2014. Special issue on hybrid intelligent methods for health technologies.

Annexes

ANNEXE A : TYPES SÉMANTIQUES

TABLE A.1 – Liste des 57 types sémantiques inclus dans cette étude part a.

Types sémantiques
CURRENCY-FRENCH
CIVILITY.MADAMEMONSIEUR-FRENCH
COUNTRY-FRENCH
ANIMAL-FRENCH
CITY-FRENCH#FRANCE
CITY-FRENCH#UNITED_KINGDOM
CONTINENT-ENGLISH
CURRENCY.REAL_CURRENCY_NUMBERS_DINAR#TUNISIA
CIVILITY.MISSSIR-ENGLISH
CIVILITY.MMEMR-FRENCH
CURRENCY.REAL_CURRENCY_NUMBERS_DOLLAR#USA
CONTINENT-FRENCH
CURRENCY.REAL_CURRENCY_NUMBERS_EURO
BLOOD_GROUP
BANK-FRENCH#FRANCE
DATE.(2J=2M=4AJ)
DATE.(2J-3M-4A)-FRENCH
FILM-FRENCH
DATE.(2J-4M-4A)-FRENCH
DISEASE-FRENCH
FIRSTNAME-FRENCH
DATE.(4A-2M-2J)
GENDER.FEMININMASCULIN-FRENCH
EMAIL
GENDER.FM

TABLE A.2 – Liste des 57 types sémantiques inclus dans cette étude part b.

Types sémantiques
DEPARTMENT_NAME_INSEE-FRENCH#FRANCE
FOREST-FRENCH#FRANCE
INSEECODE#FRANCE
TELEPHONE.MOBILENATSSESP#FRANCE
REGION_NAME_INSEE-FRENCH#FRANCE
HOSPITAL-FRENCH#FRANCE
OCEAN-FRENCH
NAME
RIVER-FRENCH#FRANCE
TELEPHONE.MOBILEINTSSESP#FRANCE
TELEPHONE.MOBILEINTSSESP#TUNISIA
SEA-FRENCH
GPS_LOCATION_LATITUDE_LONGITUDE
TELEPHONE.MOBILENATSSESP#TUNISIA
MUSEUM-FRENCH#FRANCE
HUMAIN_BONE-FRENCH
STADIUM-FRENCH#FRANCE
MANGA
URL
TEMPERATURE.FAHRENHEITFAHRENHEIT
TEMPERATURE.CELCIUSCELCIUS
TEMPERATURE.CELCIUSC
TEMPERATURE.FAHRENHEITF
WEIGHT.KILOGRAMMES-FRENCH
VEHICLE_BRAND
ZIPCODE#FRANCE
UNIVERSITY-FRENCH#FRANCE
WEIGHT.KG
WEIGHT.KILOS
WEIGHT.G
VEHICLE_REGISTRATION#FRANCE
AIRPORT-FRENCH#FRANCE

ANNEXE B : DONNÉES DE L'ENTREPRISE

Cette annexe présente l'application de la méthode développée pour la détection des types sémantiques (détaillée dans la section 3.3) sur les données de l'entreprise Synaltic.

La principale base de donnée de l'entreprise est une base traitant des associations française. La base de données est constituée principalement de deux tables contenant des informations issus de base données gouvernementales nommée RNA et Siren. Les données étant partiellement issu d'une source remplie manuellement sans contrôle certains champs sont de faible qualités et présentent de nombreuses anomalies. La table Siren compte 41 colonnes et 1475577 lignes et la table RNA 52 colonnes et 2997782 lignes

Nous avons testé notre algorithme de détection automatique de catégorie sémantiques (version "haute", utilisant la forêt d'arbre décisionnel et un seuil de 0.75) sur ces tables. Les données sont pré-traitées : mise en minuscule, suppression des espace en début et fin de chaîne de caractères et les Timestamp sont convertie en chaîne de caractères représentant la date (au même format que dans la base). Les caractéristiques sont extraites sur des échantillons de colonnes de taille 100000.

Sur table Siren l'algorithme présente un taux de bonne reconnaissance de 93% et de 88% sur la table RNA (inconnue étant ici considérée comme une catégorie) en considérant que vu le fonctionnement de l'algorithme il devrait identifier une date au format DD-MM-AAAA comme étant au format AAAA-MM-DD. Les codes postaux, INSEE et dates sont les mieux reconnu. Sur ce jeu de données les colonnes non correctement reconnu sont majoritairement très peu remplies. La principale erreur que nous avons pu constaté est une non reconnaissance des noms de ville. Celle-ci s'explique par deux facteurs. Le premier est dû à une différence entre les données de la base et les données d'apprentissage. En effet pour les communes (CITY-FRENCH#FRANCE) nous n'avons pas notés de la même manière les arrondissement que dans les bases de données de l'entreprise, cette différence de notation à une conséquence importante car les villes de Paris, Lyon et Mar-

seilles sont très représentées dans la base. Le second est dû à la trop grande proximité entre les catégories sémantiques CITY-FRENCH#FRANCE et CITY-FRENCH#UNITED_KINGDOM ce qui mène à une probabilité prédite trop faible d'appartenir à l'une ou l'autre. Un autre point soulevé lors de cette analyse est une mauvaise reconnaissance des numéros de téléphones dès que la colonne contient un mélange de numéros fixes et mobiles (car nous avons utilisé lors de l'apprentissage que des numéros de mobile)

PAPIERS SOUMIS ET EN PRÉPARATION

Liste des papiers soumis et en préparations :

• Chevallier, M., Boufarès, F., Rogovschi, N., Grozavu, N., & Clairmont, C. (2022) Détection de types sémantiques dans les données tabulaires par apprentissage automatique à l'aide de données synthétiques, BDA 2022 (Manuscrit soumis pour publication)

• Chevallier, M., Grozavu, N., Boufarès, F., Rogovschi, N., & Clairmont, C. (2022) Trade between population size and mutation rate for GAAM (genetic algorithm with aggressive mutation) for feature selection IFIP Advances In Information And Communication Technology. pp. 432-444 (2022), https ://doi.org/10.1007%252F978-3-031-08333-4_35

• Chevallier, M., Rogovschi, N., Boufarès, F., Grozavu, N., & Clairmont, C. (2022) Detecting near duplicate dataset with machine learning. International Journal Of Computer Information Systems And Industrial Management Applications vol 14 pp 374-385

• Chevallier, M., Rogovschi, N., Boufarès, F., Grozavu, N., & Clairmont, C. (2022). Detecting Near Duplicate Dataset. In Proceedings of the 13th International Conference on Soft Computing and Pattern Recognition (SoCPaR 2021) (pp. 394–403). Springer International Publishing. https ://doi.org/10.1007/978-3-030-96302-6_36

• Chevallier, M., Boufares, F., Grozavu, N., Rogovschi, N., & Clairmont, C. (2021). Near duplicate column identification : a machine learning approach. In 2021 IEEE Symposium Series on Computational Intelligence (SSCI). 2021 IEEE Symposium Series on Computational Intelligence (SSCI) IEEE. https ://doi.org/10. 1109/ssci50451.2021.9659897

• Chevallier, M., Rogovschi, N., Boufarès, F., Grozavu, N., & Clairmont, C. (2021). Techniques de génération de population initiale d'algorithmes génétiques pour la sélection de caractéristiques. Revue des Nouvelles Technologies de l'Information RNTI (Manuscrit soumis pour publication)

• Chevallier, M (2021) Management of fuzzy-duplicate for data integration with machine learning. Workshop on Machine Learning and Applications. International Conference on Electronics, Communications and Computing (ECCO 2021), Chisinau.

• Chevallier, M., Rogovschi, N., Boufarès, F., Grozavu, N., & Clairmont, C. (2021). Techniques de génération de population initiale d'algorithmes génétiques pour la sélection de caractéristiques Actes de la 9e Conférence Internationale Francophone sur la Science des Données (pp 25-35). HAL.

• Chevallier, M., Rogovschi, N., Boufarès, F., Grozavu, N., & Clairmont, C. (2021). Seeding Initial Population, in Genetic Algorithm for Features Selection. In Advances in Intelligent Systems and Computing (pp. 572–582). Springer International Publishing. https ://doi.org/10.1007/978-3-030-73689-7_55