UNIVERSITÉ DU QUÉBEC À MONTRÉAL

OPTIMISATION DES HYPERPARAMÈTRES POUR LA CLASSIFICATION DES VIRUS

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR
FATEN BELARBI

JUILLET 2019

REMERCIEMENTS

Nulle œuvre n'est plus exaltante que celle réalisée avec le soutien moral et financier des personnes qui nous sont proches. Afin d'être reconnaissant envers ceux qui m'ont appuyé et encouragé à effectuer ce travail de recherche, je remercie :

Mon directeur de recherche Monsieur Abdoulaye Baniré Diallo, professeur et chef du laboratoire de Bioinformatique, de m'avoir orienté et encouragé au cours de mon travail, également pour sa gentillesse, ses compétences et sa disponibilité continue malgré toutes ses journées surchargées. Il a su me faire partager ses nombreuses connaissances, sa vision toujours claire et synthétique. Ma sincère gratitude et un grand respect vous sont personnellement adressés, pour m'avoir fait confiance, pour m'avoir beaucoup appris.

Tous les membres du laboratoire de bioinformatique auquel j'ai l'honneur d'appartenir, merci en particulier et très spécialement à Amine M. Remita et Golrokh Kiani qui par leurs encouragements, leur aide, leurs conseils et leurs critiques, ont contribué à la réalisation de ce travail. Je remercie aussi Ahmed Halioui, Abou Abdallah Malick Diouara, Bruno Daigle, Steve Ataky, Julie Chao-Jung et Dylan Lebatteux pour leur aide, leurs conseils et leur bonne humeur.

Mon père, qui a œuvré pour ma réussite, de par son amour, son soutien, tous les sacrifices consentis et ses précieux conseils, pour toute son assistance et sa présence dans ma vie. Reçois à travers ce travail aussi modeste soit-il, l'expression de mes sentiments et de mon éternelle gratitude.

Ma mère, qui peut être fière de trouver ici le résultat de longues années de sacrifices pour m'aider à avancer dans la vie. Puisse Dieu faire en sorte que ce travail porte son fruit. Merci pour les valeurs nobles, l'éducation et le soutien permanent venu de toi.

Mon frère et ma sœur, qui n'ont cessé d'être pour moi une source de courage et

de soutien, un soutien sans lequel le succès ne serait possible.

Je voudrais aussi exprimer ma reconnaissance envers tous ceux qui par leurs encouragements, leur support moral et/ou intellectuel ont contribué à la réalisation de ce travail.

Enfin, je tiens à témoigner toute ma gratitude à Olivier, Azer, Oussama et Wassim qui m'ont toujours encouragé et soutenu durant mes études mais également supporté lors des moments difficiles.

À toutes et tous, un grand merci !

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ALGORITHMES

LISTE DES ABBRÉVIATIONS, DES SIGLES ET DES ACRONYMES

ADN Acide désoxyribonucléique

ARN Acide ribonucléique

k-NN k plus proches voisins

SMBO Optimisation basée sur un modèle séquentiel

SVM Machine à vecteurs de support

RÉSUMÉ

La plupart des algorithmes d'apprentissage automatique dépendent étroitement de la configuration manuelle de leurs hyperparamètres pour assurer une évaluation fiable de chaque ensemble de données particulier. Le choix des valeurs de ces hyperparamètres est rarement mis en valeur, il est présenté comme accessoire à l'algorithme mais ce choix est souvent étroitement lié à la qualité des résultats obtenus.

Généralement, chaque fois qu'on change le problème, les données ou l'algorithme d'apprentissage, il devient nécessaire de réassigner les valeurs des hyperparamètres sans qu'il y ait une science exacte pour nous guider dans ce choix. Ce processus dépend de l'expérience personnelle de l'expert et repose beaucoup sur l'intuition. Il est assez difficile à le quantifier ou à le décrire. Ainsi étant donné que la performance d'une technique dépend de plusieurs choix, il est parfois difficile de trancher si elle est vraiment meilleure ou simplement mieux réglée.

Dans ce travail, nous nous concentrons sur une tâche fondamentale dans l'apprentissage automatique : l'optimisation de la classification dans un ensemble ouvert. Nous travaillons sur des données de séquences virales comme un domaine d'application. Nous utilisons Galaxy-X, une approche de classification pour les problèmes de classification dans un espace ouvert comme exemple à optimiser. Pour chaque classe de l'ensemble d'apprentissage, cet outil crée une hypersphère de délimitation avec un rayon minimal qui englobe sa distribution en contenant toutes ses instances. De cette manière, il est capable de distinguer des instances appartenant à ces classes précédemment vues de celles qui sont nouvelles ou inconnues.

Notre approche consiste à utiliser plusieurs méthodes d'optimisation et à comparer leurs résultats. Les méthodes présentées dans ce travail sont : la recherche exhaustive, l'algorithme de descente, l'optimisation bayésienne, ainsi qu'un nouvel algorithme que nous présentons : la recherche paramètre par paramètre. Les résultats expérimentaux sur les ensembles de données de référence et sur des données virales nous ont permis de faire une comparaison entre la performance de plusieurs algorithmes d'optimisation.

Mots clés : réglage automatique des hyperparamètres, classification dans un ensemble ouvert, optimisation des hyperparamètres, optimisation bayésienne, re-

cherche exhaustive, algorithme de descente, apprentissage automatique automatisé, apprentissage autonome.

INTRODUCTION

Les virus sont une cause importante de problèmes de santé dans le monde, en particulier dans les pays en développement. Les populations humaines sont exposées à différents agents pathogènes viraux, dont beaucoup se présentent sous forme de foyers. Dans de telles situations, l'identification de nouveaux virus est primordiale pour décider des stratégies de préventions et de traitements.

Au cours du vingtième siècle, des méthodes de détection, de caractérisation et de classification taxonomique des virus ont été mises au point. Elles ont permis de découvrir un certain nombre de virus importants, de prévenir les infections virales et de les traiter. Vers la fin des années cinquante, on pensait généralement que la plupart des virus pathogènes pour l'homme avaient été découverts, mais l'apparition d'un certain nombre de virus inconnus auparavant, tel que le virus Coronavirus : Ébola, au cours de la dernière partie de ce siècle, a fortement remis en question cette croyance (Bichaud *et al.*, 2014).

Outre les épidémies naturelles, le risque d'utilisation d'agents pathogènes, notamment de virus mortels, comme armes biologiques et agents du bioterrorisme a également augmenté ces dernières années (Bronze *et al.*, 2002). Exceptionnellement diversifiés en termes d'étiologie, de morphologie, de type d'acide nucléique et d'informations de séquence, de manifestations cliniques, etc., la détection et l'identification rapides des virus constituent un défi de taille pour les chercheurs cliniques. Néanmoins, lors d'épidémies naturelles ou délibérées, l'identification et la caractérisation des virus dans les échantillons cliniques sont essentielles pour faciliter les stratégies de prévention et de quarantaine, mettre en oeuvre des outils

de diagnostic spécifiques et définir des stratégies de traitement explicites.

La classification des virus implique de nommer et de placer les virus dans un système taxonomique. À l'instar des systèmes de classification relativement cohérents observés pour les organismes cellulaires, la classification du virus fait l'objet de débats et de propositions en cours. Ceci est en grande partie dû à la nature pseudo-vivante des virus, qui ne sont pas encore considérés définitivement vivants ou non-vivants. En tant que tels, ils ne s'inscrivent pas parfaitement dans le système de classification biologique établi en place pour les organismes cellulaires, tels que les végétaux et les animaux, pour plusieurs raisons.

La classification des virus repose principalement sur leurs caractéristiques phénotypiques notamment la morphologie, le type d'acide nucléique, le mode de réplication, les organismes hôtes et le type de maladies qu'ils provoquent. David Baltimore, biologiste lauréat du prix Nobel 1975, a mis au point le système de classification de Baltimore, qui classe les virus dans l'un des sept groupes. Ces groupes sont désignés par des chiffres romains et séparent les virus en fonction de leur mode de réplication et de leur type de génome. Cette méthode générale de classification est accompagnée de conventions de dénominations spécifiques et de directives de classifications supplémentaires définies par le Comité International de la taxonomie des Virus (Yu *et al.*, 2013).

Récemment, l'émergence et la disponibilité commerciale des séquenceurs de nouvelle génération ont complètement changé le domaine de la découverte de virus. Ces plateformes de séquençage massivement parallèles peuvent séquencer un mélange de matériel génétique à partir d'un mélange très hétérogène et avec une grande précision. De plus, ces plateformes fonctionnent indépendamment de la séquence, ce qui constitue en fait des outils idéaux pour la découverte de virus.

Toutes les plateformes de séquenceurs de nouvelle génération progressent vers la

possibilité de séquencer des fragments d'ADN plus longs et de générer un volume encore plus important d'ensembles de données (Escalante *et al.*, 2014). Pour analyser de tels ensembles de données, des installations informatiques exceptionnellement volumineuses sont également nécessaires, ce qui a entièrement révolutionné le domaine de la bioinformatique (Mardis, 2008, Henson *et al.*, 2012).

Ainsi, les méthodes automatisées de classification des virus en se basant strictement sur le génotype des virus sont nécessaires pour simplifier le travail des taxonomistes. Différentes techniques d'apprentissage automatique peuvent être adaptées pour résoudre ce problème sauf que le choix de la bonne technique et la mise au point de ses hyperparamètres sont des tâches cruciales qui auront un impact direct sur la qualité des prédictions. Cependant, cette décision n'est pas une tâche facile, car le nombre de choix est généralement illimité.

Nous proposons en effet, une approche de classification virale qui, en se basant sur un algorithme de classification dans un environnement ouvert, vise à automatiser le choix optimal des hyperparamètres de cet algorithme pour assurer une meilleure performance pour n'importe quel choix de données.

Cette classification se résume en trois phases : l'apprentissage des données, l'optimisation des hyperparamètres et l'entraînement final du modèle.

Ainsi, le mémoire s'articule autour des chapitres suivants :

— Chapitre I : Cadre théorique : Des notions de biologie et d'apprentissage automatique seront présentées pour bien limiter la problématique de recherche.
— Chapitre II : État de l'art : nous présentons les différentes méthodes d'optimisation des hyperparamètres ainsi que l'algorithme Galaxy-X que nous avons adopté comme exemple, à généraliser et à optimiser.

— Chapitre III : Problématique : Nous présentons la problématique dans ce mémoire et les différents objectifs que nous souhaitons réaliser.

— Chapitre IV : Matériels et méthodes : la classification est décrite sous l'angle de l'apprentissage puis, plus contextuel, du point de vue des méthodes d'optimisation. La discussion portera par la suite, sur la façon dont ces méthodes vont être appliquées sur Galaxy-X.

— Chapitre V : Résultats : Les résultats seront révélés à ce stade, notamment une comparaison entre les scores de classification de chaque méthode d'optimisation.

— Conclusion : Les principales réalisations seront reprises en guise de conclusion.

CHAPITRE I

CADRE THÉORIQUE : NOTIONS DE BIOLOGIE ET D'APPRENTISSAGE AUTOMATIQUE

Les sciences biologiques ont été révolutionnées par les informations obtenues à partir des technologies de séquençage de l'ADN au cours des deux dernières décennies. Les énormes quantités de données qui peuvent maintenant être produites dans un court laps de temps et à un coût relativement bas nécessitent des algorithmes avancés pour bien profiter de ces informations. Compte tenu de la quantité de données à traiter, il est important que ces algorithmes soient non seulement très précis, mais également rapides.

1.1 Notions de biologie

Les difficultés liées à la classification des virus ont été amplifiées par les récents progrès en matière de séquençage et de méta-génomique de nouvelles générations.

Pour faire face au volume croissant des nouvelles données de séquençage virale, il a été proposé que le Comité international de la taxonomie des virus change la classification officielle des virus en se basant uniquement sur des informations de séquence de génome de virus et méta-génomique (Simmonds *et al.*, 2017).

1.1.1 Virus

Les virus sont de petits agents infectieux intracellulaires qui, par définition, contiennent un génome d'ARN ou d'ADN entouré par une couche protectrice de protéines codées par le virus et capables de se répliquer à l'intérieur des cellules vivantes d'un organisme hôte. Les virus peuvent être considérés comme des éléments génétiques mobiles, très probablement d'origine cellulaire.

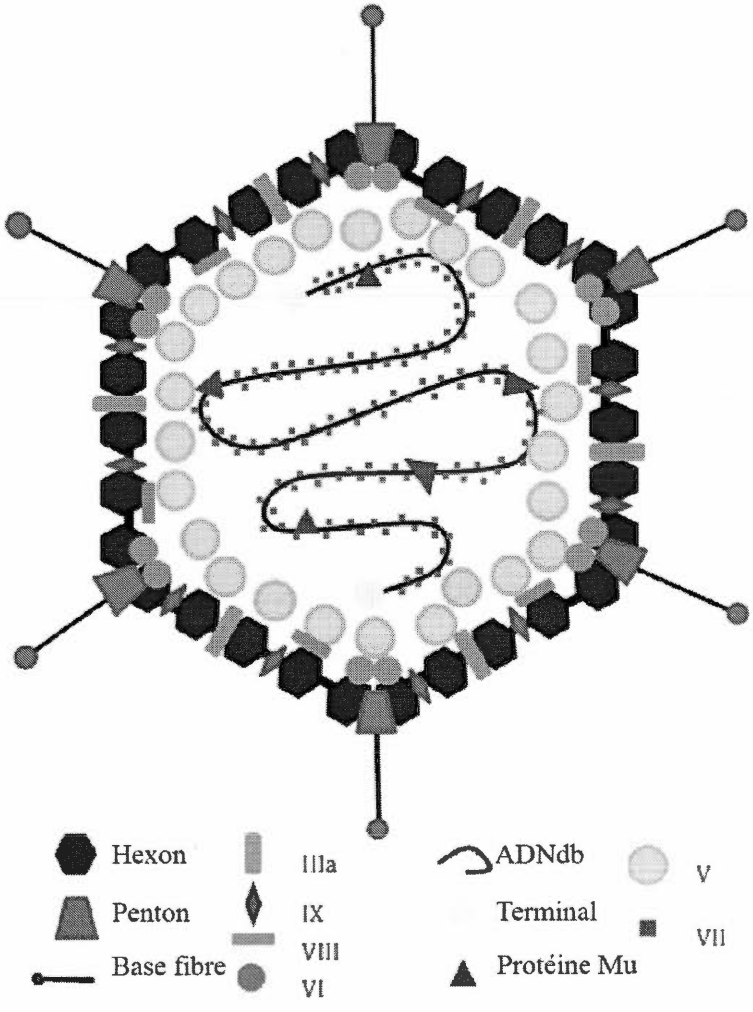


Figure 1.1: Structure d'un adénovirus
(Waye et Sing, 2010) adaptée

Pour se propager, les virus dépendent d'une cellule hôte fournissant les mécanismes métaboliques et biosynthétiques complexes nécessaires à la production des protéines à partir du génome viral. Les cellules hôtes peuvent être eucaryotes ou procaryotes. La forme complète infectieuse d'un virus s'appelle un virion et leur fonction principale est de délivrer son génome d'ADN ou d'ARN dans la cellule hôte, afin que le génome puisse être transcrit et traduit par la cellule hôte et enfin pour qu'ils s'y répliquent.

Le génome viral, est emballé dans une capside de protéine (Figure 1.1). La protéine associée aux acides nucléiques, appelée nucléoprotéines, avec le génome, forment la nucléocapside (Gelderblom, 1996).

Nous devons la première définition moléculaire d'un virus à André Lwoff (1957) : "Les virus sont infectieux, potentiellement pathogéniques, des entités protéiques ne possédant qu'un seul type d'acide nucléique, qui sont reproduits à partir de leur matériel génétique, ne sont pas capables de pousser ou de se diviser et sont dépourvus de système Lipman" (le système Lipman est l'ensemble des enzymes permettant de transférer de l'énergie vers des liaisons chimiques).

Ce dernier point fut rapidement corrigé par Salvador Luria qui démontre que les virus pouvaient posséder les deux types d'acides nucléiques. Bien sûr cette définition évolua au fil du temps et des découvertes si bien qu'à l'heure actuelle, l'une des définitions les plus simples et les plus souples que l'on puisse donner est celle proposée par Luis Villareal : "Un virus est un parasite génétique qui utilise les systèmes cellulaires pour sa propre réplication".

Selon Lwoff, un virus serait donc un parasite génétique ne pouvant s'autorépliquer que dans son hôte cellulaire par le détournement de la machinerie biochimique de ce dernier, et ne possédant qu'un seul type d'acide nucléique (ADN ou ARN).

L'avantage de cette définition est qu'elle ne fait pas référence à l'identité moléculaire du virus. Elle ne fait également pas référence aux gènes viraux, ou aux rôles de ces derniers dans le cycle de réplication du virus.

Cette définition permet de plus d'inclure aussi bien les virus traditionnels qui se transmettent par voie extracellulaire (et qui fabriquent par conséquent des virions ayant une structure moléculaire spécifique) que ceux qui se transmettent à travers le génome de leurs hôtes, ou par d'autres moyens invisibles, ce qui inclut les virus défectifs (des virus ne pouvant se répliquer seuls et ayant besoin de l'aide d'un autre virus, comme par exemple le virus de l'hépatite D qui ne peut se propager qu'en présence de l'hépatite B).

1.1.2 Virologie et classification des virus

La classification et la division des virus en ordres, familles, genres et espèces permet de mieux comprendre leurs propriétés biologiques.

La diversité virale est toutefois bien supérieure à celle d'autres organismes, avec des différences majeures de leur matériel génétique (ARN ou ADN), des configurations doubles ou simples brins et de l'orientation de leurs gènes codés. De plus, les génomes viraux peuvent être répartis sur plusieurs segments, parfois emballés ensemble dans un virion, ou souvent dans des particules virales séparées, qui sont tous nécessaires pour infecter une cellule afin que la réplication puisse se produire.

Les génomes viraux sont de différentes tailles, reflétant divers mécanismes de réplication et d'interactions, ainsi que la complexité structurelle variable de leurs virions. Les plus petits génomes de virus vont de moins de 2 000 bases, contenant deux gènes, à 2,5 millions de paires de bases, contenant plus de 2 500 gènes (Abergel *et al.*, 2015).

Historiquement, les virus étaient classés en fonction des maladies dont ils étaient

responsables. Ce type de classification est cependant hasardeux car un virus peut causer un ou plusieurs types de maladies, voir aucune. De plus, les virus peuvent provoquer des maladies différentes mais entraînent les mêmes symptômes cliniques, faussant ainsi ce type de classification.

L'utilisation de cette méthode fut définitivement abandonnée avec l'avènement de la biologie moléculaire. Désormais, les virus sont le plus souvent classés suivant la classification de Baltimore en se basant sur :

- L'acide nucléique constituant leur génome (ADN, ARN, simple ou double brin) ;
- Leur stratégie réplicative (nucléaire ou cytoplasmique).

Sept ordres viraux existent dans cette classification (figure 1.2) :

- ADNdb : génomes viraux à ADN double brin ;
- ADNsb : génomes viraux à ADN simple brin ;
- ARNdb, génomes viraux à ARN double brin ;
- ARNsb- : génomes viraux à ARN anti-sens simple brin ;
- ARNsb+ : génomes viraux à ARN sens simple brin ;
- ARNsb-RT : génomes viraux à ARN sens simple brin ayant une étape intermédiaire à ADN ;
- ADNdb-RT : génomes viraux à ADN double brin avec une phase intermédiaire à ARN.

En 2014, le Comité International sur la Taxonomie des Virus a décrit :

- 7 ordres ;
- 104 familles ;
- 23 sous-familles ;
- 505 genres ;
- 3186 espèces virales.

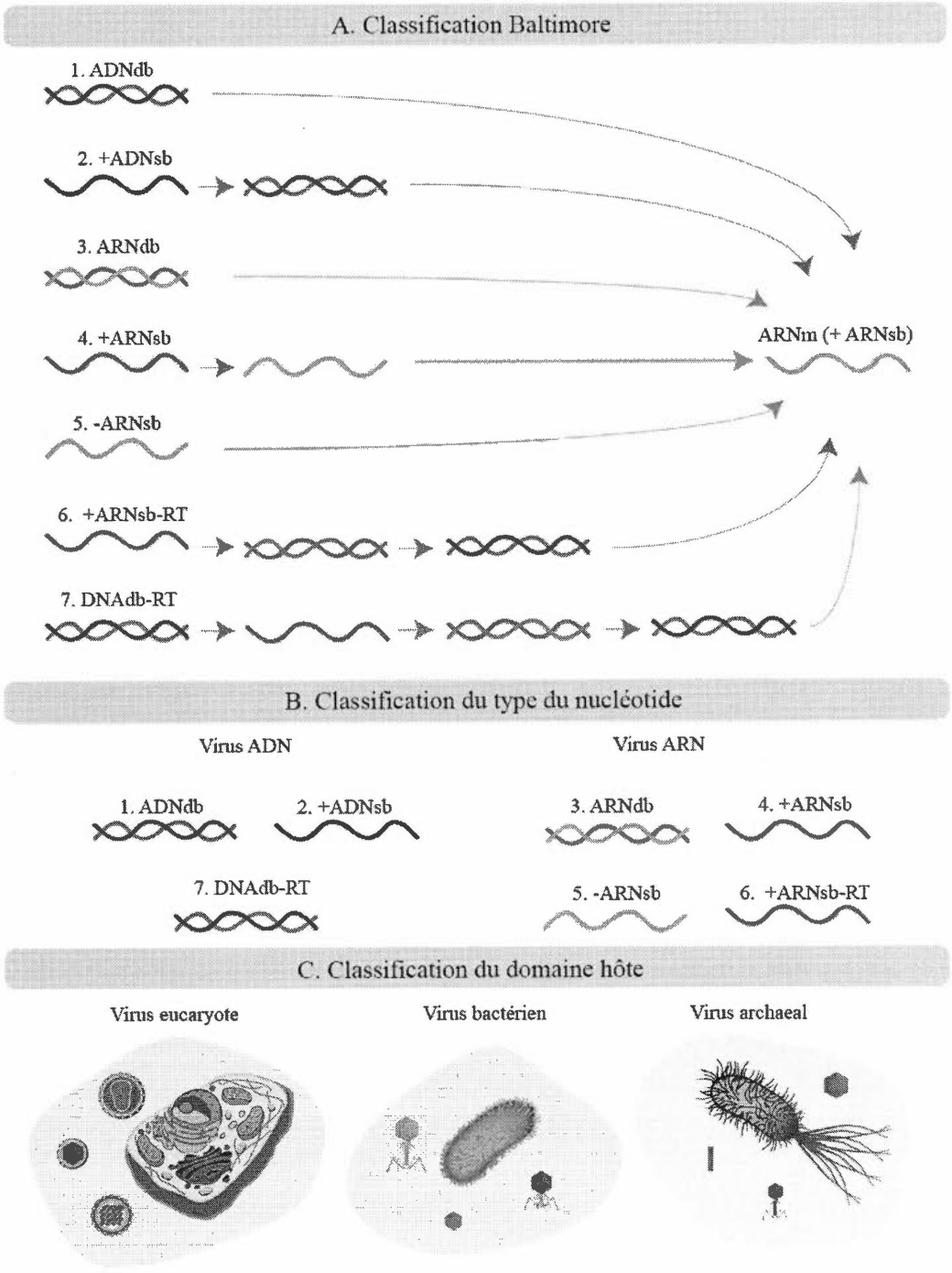


Figure 1.2: Schémas de plusieurs systèmes de classification virale *(Mahmoudabadi et Phillips, 2018) adaptée*

La manipulation et l'analyse des séquences d'ADN restent l'une des tâches principales de la bioinformatique. Ce sujet est généralement divisé en deux parties :

— La génomique fonctionnelle, qui cherche à déterminer le rôle de la séquence dans la cellule vivante, soit en tant qu'unité transcrite et traduite (c'est-à-dire une protéine dont la description pourrait en comprendre la connaissance, la structure et les interactions possibles), ou en tant que motif régulateur, que ce soit en tant que site promoteur ou en tant que séquence courte transcrite en un morceau de petit ARN interférant (Amiri et Dinov, 2016) ;

— La génomique comparative, dans laquelle les séquences d'organismes différents, voire d'individus différents, sont comparées afin de déterminer les ancêtres et les corrélations avec la maladie. Il est clair que la comparaison de séquences inconnues avec des séquences connues peut également aider à élucider leur fonction (Gelderblom, 1996).

La classification des séquences a un large éventail d'applications telles que l'analyse génomique, la recherche d'informations, l'informatique de santé, les finances et la détection d'anormalités (Simmonds et Aiewsakun, 2018).

Afin de réaliser la classification virale basée sur la séquence du génome, des méthodes de comparaison de séquences sont nécessaires pour établir la relation entre différents virus. En général, la classification basée sur la séquence du génome peut être divisée en deux types : méthodes basées sur l'alignement et sans alignement.

1.1.2.1 Méthodes basées sur l'alignement

Les méthodes basées sur l'alignement constituent l'approche traditionnelle de la comparaison des séquences et classifient les virus en fonction du degré de similitude de leurs codes génomiques. Ces méthodes identifient d'abord les régions

conservatrices dans les séquences du génome, puis les alignent par insertion, délétion et mutation, et finalement dérivent des mesures de distance entre les génomes en utilisant les scores d'alignement.

Il existe de nombreuses techniques de ce type, qui diffèrent principalement par la manière dont les séquences sont alignées et/ou par les scores obtenus (Durbin *et al.*, 1998, Waterman, 1995). Par exemple, l'alignement peut être effectué toutes les deux séquences (Muhire *et al.*, 2013, Bao *et al.*, 2014, Muhire *et al.*, 2014) ou entre plusieurs simultanément (Katoh *et al.*, 2002, Edgar, 2004, Larkin *et al.*, 2007) ; l'alignement peut être effectué uniquement sur certaines structures locales de séquences (Altschul *et al.*, 1990, 1997, Bao *et al.*, 2014) ou sur la structure globale dans son ensemble (Needleman et Wunsch, 1970, Bao *et al.*, 2014). Une large gamme de systèmes de notation a également été proposée, tels que les matrices de similarités PAM (Dayhoff et Schwartz, 1978) et BLOSUM (Henikoff et Henikoff, 1992).

Ces méthodes fonctionnent bien pour des ensembles de données relativement petits, constitués de séquences similaires, mais peuvent présenter des limitations à la fois informatiques et fondamentales pour des ensembles de données plus volumineux comportant diverses séquences.

En termes de charge de calcul, un alignement optimal des séquences peut être irréalisable pour les grands corpus de séquences produites par les technologies de séquençage de nouvelle génération (Liu *et al.*, 2012, Just, 2001). Basé sur l'alignement, ces méthodes (par exemple, (Altschul *et al.*, 1990, Edgar, 2004)) nécessitent généralement une grande complexité temporelle et spatiale qui dépend de la longueur moyenne des séquences.

Des méthodes plus efficaces (par exemple (Delcher *et al.*, 1999, Kaur et Sohi, 2017) ont été développées à des fins spécialisées, mais supposent généralement des

propriétés spécifiques pour les séquences alignées. En termes de fondamentaux virologiques, les hypothèses évolutives guidant les procédures d'alignement et de notation peuvent ne pas refléter la phylogénie, tendant à trop insister sur l'importance de la similarité de séquence tout en négligeant l'importance de la similitude fonctionnelle (Lynch, 2002, Zhang *et al.*, 2002).

Par ailleurs, les méthodes d'attribution de scores supposent la linéarité de la procédure évolutive, qui se déroule en fait à différentes échelles simultanément (Attwood, 2000). De plus, en raison de l'absence de représentation des caractéristiques, ils ne peuvent être combinés qu'à des classificateurs basés sur la distance, ce qui restreint l'application de technologies potentiellement plus sophistiquées et plus puissantes techniques d'apprentissage automatique.

1.1.2.2 Méthodes sans alignement

Au contraire, les méthodes sans alignement, au centre de cette thèse, classent les virus en fonction de la mesure dans laquelle les caractéristiques des différentes séquences sont similaires.

Au lieu d'aligner des séquences ou d'obtenir des scores de similarité, elles associent d'abord une séquence du génome du virus sur un point de l'espace des caractéristiques, où la distance entre les entités reflète la distance entre les séquences originales, puis classent le virus dans cet espace.

En se basant uniquement sur l'analyse de la séquence du génome, des méthodes sans alignement souffrent des mêmes inconvénients que les méthodes basées sur l'alignement, en l'absence d'un aperçu des fonctionnalités de leurs virus. Cependant, les premiers améliorent ce dernier sous plusieurs aspects :

— Tout d'abord, aucun alignement n'est nécessaire, les ressources biologiques associées, les connaissances requises pour informer le processus d'aligne-

ment ne sont pas nécessaires, ce qui peut être un avantage dans les situations où seule la séquence est connue ;

— Deuxièmement, ils peuvent faire face à des séquences très diverses où un alignement fiable est impossible (Vinga et Almeida, 2003) ;

— Troisièmement, ils peuvent gérer plus efficacement de grands ensembles de données car toutes les séquences sont représentées dans un format fixe sous forme de points dans un espace de fonctions.

En outre, l'utilisation de fonctionnalités permet l'application d'un plus large éventail de techniques d'apprentissage automatique, comme le classificateur k-NN (Yu *et al.*, 2010, Hernandez et Yang, 2016), le classificateur basé sur des règles d'association (She *et al.*, 2003), SVM (Leslie *et al.*, 2004) et les réseaux de neurones artificiels (Yu *et al.*, 2013).

Bien que les virologistes se sont traditionnellement concentrés sur les virus qui causent des maladies chez l'homme, les animaux domestiques et les cultures, les récents progrès en matière de séquençage métagénomique, en particulier séquençage à haut débit d'échantillons environnementaux, ont révélé un virome incroyablement grand partout dans la biosphère. Il existe au moins 1031 particules de virus dans le monde à un moment donné, dans la plupart des environnements, y compris les habitats marins et d'eau douce, le tractus gastrointestinal métazoaire, dans lesquels le nombre de particules de virus détectables dépasse de 10 à 100 fois le nombre de cellules (Simmonds *et al.*, 2015).

1.2 Notions d'apprentissage auromatique

1.2.1 Définitions

L'apprentissage automatique est une branche de l'intelligence artificielle, dédiée à l'étude de modèles et d'algorithmes capables d'apprendre automatiquement. Il

permet de résoudre une variété de tâches en se basant sur des données existantes, avec un minimum d'interaction humaine.

De plus en plus, les connaissances accumulées du monde sont stockées sous format électronique. Une énorme quantité de documents, d'images, de sons, de vidéos et de discussions, est disponible grâce à l'internet.

L'apprentissage automatique est au croisement des statistiques, des probabilités, du calcul, de l'algèbre linéaire et de l'optimisation numérique, couplé à la biologie et à la psychologie. Main en main avec la croissance de la puissance de calcul et la quantité de données disponibles, la performance des systèmes d'apprentissage automatique ne cesse d'augmenter.

L'apprentissage automatique est une application actuelle de l'intelligence artificielle basée sur l'idée de pouvoir donner aux machines un accès aux données et les laisser apprendre par elles-mêmes. Les techniques d'apprentissage automatique sont basées sur des algorithmes de calcul capables d'extraire des règles des données d'apprentissage et de les appliquer aux données de test (prédiction) sans être explicitement programmé (Bishop, 2006). Entre autres utilisations, ces techniques peuvent automatiser les tâches manuelles pour aider les experts humains à analyser des ensembles de données volumineux et complexes.

Les techniques d'apprentissage automatique peuvent être classées de différentes manières. Par exemple, selon que chaque échantillon des données d'entraînement se voie attribuer une étiquette de classe, elles peuvent être divisées en méthodes supervisées, non supervisées et semisupervisées. Elles peuvent également être divisées en fonction des matrices de distance ou des vecteurs de caractéristiques, qui sont requis comme entrée. Enfin, elles peuvent être divisées en méthodes non hiérarchiques et hiérarchiques selon la relation entre les classes.

La conception, l'analyse, le développement et l'implémentation de méthodes d'apprentissage automatique constituent un processus d'apprentissage d'un ensemble de règles à partir d'instances (ensemble d'entraînement) ou, plus généralement, de la création d'un classificateur ou modèle pouvant être utilisé pour généraliser à partir de nouvelles instances.

La première étape consiste à collecter le jeu de données. L'expert doit suggérer quels champs (attributs et caractéristiques) sont les plus informatifs. Si un expert n'est pas disponible, la méthode la plus simple est la 'force brute' : mesurer tout ce qui est disponible dans l'espoir que les caractéristiques informatives et pertinentes peuvent être isolées.

Cependant, un jeu de données collecté par la méthode de la 'force brute' ne convient pas directement à l'induction. Il contient dans la plupart des cas des valeurs de bruit et de caractéristiques manquantes et nécessite par conséquent un pré-traitement important (Zhang *et al.*, 2002).

Chaque instance d'un jeu de données utilisé par des algorithmes d'apprentissage automatique est représentée à l'aide du même jeu d'attributs. Les fonctionnalités peuvent être continues, catégoriques ou binaires. Si les instances sont indiquées avec des étiquettes connues (la sortie correcte correspondante), alors l'apprentissage est appelé supervisé, contrairement à l'apprentissage non supervisé où les instances ne sont pas étiquetées.

1.2.2 Catégories de l'apprentissage automatique

1.2.2.1 Approches supervisées vs non supervisées vs semi-supervisées

Ces approches permettent de construire un modèle de la distribution des étiquettes de classe. Le classificateur résultant est ensuite utilisé pour affecter des étiquettes de classe aux instances de test où les valeurs des attributs de prédicteur sont

connues, mais la valeur de l'étiquette de classe est inconnue.

Les techniques d'apprentissage automatique peuvent être divisées en trois classes principales basées sur la présence d'étiquettes lors de la construction de modèles :

— Apprentissage supervisé, où les informations d'étiquetage des données connues guident le processus d'apprentissage ;
— Apprentissage non supervisé, où les règles sont apprises par les propriétés intrinsèques des données sans l'aide d'informations d'étiquetage ;
— Apprentissage semi-supervisé, qui combine des données étiquetées et non étiquetées au cours de l'apprentissage et de la prédiction.

Appliquées à la taxonomie des virus, les techniques d'apprentissage supervisé (telles que la classification) peuvent être utilisées pour prédire les taxons, étiquettes d'un nouveau virus.

D'autre part, les techniques d'apprentissage non supervisé (telles que le regroupement) peuvent être utilisées pour explorer des groupes de virus naturels et construire des arbres phylogénétiques (Tomović *et al.*, 2006).

Le troisième type, les techniques semi-supervisées, peut être utilisé pour améliorer la classification ou le regroupement dans les situations où des étiquettes de taxons sont disponibles pour un petit nombre d'échantillons mais sont manquantes pour les autres.

1.2.2.2 Approches basées sur la distance vs approches basées sur les attributs

Les méthodes d'apprentissage automatique peuvent être divisées en trois types, en fonction des mécanismes qu'elles utilisent pour séparer les données et former les classes :

— Le premier type est basé sur la distance, telles que k-plus proches voisins

(k-NN) et k-moyennes (Cover et Hart, 1967). Il ne nécessite pas des représentations caractéristiques des échantillons pour autant que leur distance par paire puisse être obtenue. Ces méthodes prennent la matrice de distance calculée à partir de tous les échantillons, comme saisir et regrouper les personnes proches ;

— Le second type est relatif à des méthodes basées sur les fonctionnalités, telles que les arbres de décision (Safavian et Landgrebe, 1991) et les forêts aléatoires (Ho, 1995). Contrairement aux méthodes basées sur la distance, pour ces méthodes, les représentations des données d'origine sont obligatoires. Ces méthodes prennent les caractéristiques d'un échantillon en entrée et attribuent des étiquettes en examinant le pouvoir discriminant des variables individuelles constituant chaque caractéristique ;

— Le troisième type combine les deux précédents, par exemple les machines à vecteurs de support (Scholkopf et Smola, 2001) et d'autres méthodes de noyau. Il nécessite à la fois une définition de mesure de distance spécifique et une fonctionnalité de représentation. Les méthodes du noyau prennent les entités en entrée et les transforment implicitement dans un espace de grande dimension à travers une fonction du noyau, qui joue un rôle similaire, à une mesure de distance.

Les avantages d'une telle division peuvent être clairs lorsque nous discutons des méthodes d'analyse de séquences basées sur l'alignement et sans alignement : le manque d'attributs limite les méthodes basées sur l'alignement comparées aux méthodes basées sur la distance uniquement, alors que la disponibilité de fonctionnalités permet à une large gamme de classificateurs d'être combinés avec ces méthodes. Une brève discussion sur les méthodes de séquençage basées sur la distance et leurs différentes fonctionnalités pour la classification peuvent être trouvée dans (Xing *et al.*, 2010).

1.2.2.3 Approches non-hiérarchiques vs hiérarchiques

Les méthodes d'apprentissage automatique peuvent également être divisées en approches non hiérarchiques et hiérarchiques selon que les différentes classes d'échantillons ont ou non une relation hiérarchique entre eux. Ceci est décidé en calculant la dissimilarité entre eux. La plupart des techniques d'apprentissage automatique ne sont pas conçues à l'origine pour traiter explicitement les relations hiérarchiques et ne supposent généralement aucune relation hiérarchique entre différentes classes.

Les approches hiérarchiques sont généralement développées en incorporant des informations hiérarchiques dans leurs équivalents non hiérarchiques, que nous appelons 'classificateurs de base' dans le contexte de la classification hiérarchique (han *et al.*, 2011).

1.2.2.4 Classification dans un espace ouvert et détection des données aberrantes

Une valeur aberrante est une donnée très différent des données restantes. Hawkins a formellement défini la notion de valeur aberrante comme suit : "Une valeur aberrante est une observation qui s'écarte tellement des autres observations qu'elle suscite des soupçons qu'elle a été générée par un mécanisme différent." (Hawkins, 1980)

Les valeurs aberrantes sont également appelées discordantes, déviantes ou anomalies dans la littérature relative à l'exploration de données et à la statistique. Dans la plupart des applications, les données sont créées par un ou plusieurs processus générateurs, qui peuvent refléter une activité dans le système ou des observations collectées sur des entités.

Lorsque le processus de génération se comporte de manière inhabituelle, il en ré-

sulte une création de valeurs aberrantes. Par conséquent, une valeur aberrante contient souvent des informations utiles sur les caractéristiques anormales des systèmes et des entités, qui ont une incidence sur le processus de génération de données. La reconnaissance de ces caractéristiques inhabituelles fournit des informations utiles et spécifiques à l'application.

La sortie d'un algorithme de détection de valeurs aberrantes peut être de deux types :

— L'algorithme de détection de la plupart des valeurs aberrantes génère un score relatif au niveau de «valeur aberrante» d'un point de données. Ceci peut être utilisé pour déterminer un classement des points de données en fonction de leur tendance aux valeurs aberrantes. Il s'agit d'une forme de sortie très générale, qui conserve toutes les informations fournies par un algorithme particulier, mais ne fournit pas une idées concise sur le petit nombre de points de données qui doivent être considérés comme des valeurs aberrantes.

— Un deuxième type de sortie est une étiquette binaire indiquant si un point de données est une valeur aberrante ou non. Certains algorithmes peuvent directement renvoyer des étiquettes binaires, mais les scores des valeurs aberrantes peuvent également être convertis en étiquettes binaires. Cela se fait généralement en imposant des seuils aux scores aberrants, en fonction de leur distribution statistique. Un étiquetage binaire contient moins d'informations qu'un mécanisme d'attribution de scores, mais c'est le résultat final qui est souvent nécessaire pour la prise de décision dans des applications pratiques.

Dans les applications réelles, les données peuvent être intégrées à une quantité importante de bruit, qui peut ne pas intéresser l'analyste. Ce sont généralement

les déviations significativement intéressantes qui sont importantes. Les techniques de détection de valeurs aberrantes supervisées sont généralement beaucoup plus efficaces dans de nombreux scénarios spécifiques à une application, parce que les caractéristiques des exemples précédents peuvent être utilisées pour orienter le processus de recherche vers des valeurs aberrantes plus pertinentes.

Aussi les méthodes de classification existantes sont conçues pour classer des instances inconnues dans un ensemble de classes de formation connue auparavant. Une telle classification prend la forme d'une prédiction dans un ensemble fermé de classes.

Cependant, un scénario plus réaliste qui convient aux applications du monde réel consiste à envisager la possibilité de rencontrer des instances n'appartenant à aucune des classes de formation, c'est-à-dire une classification ouverte. Dans une telle situation, les classificateurs fermés existants attribueront une étiquette d'apprentissage à ces instances, ce qui entraînera une classification erronée.

En raison de la croissance de la collecte de données dans de nombreuses applications du monde réel, les données de formation ne représentent qu'une vue partielle du domaine et ne contiennent donc pas d'exemples de formation pour toutes les classes possibles. Dans ce cas, le classificateur peut être confronté à des observations n'appartenant à aucune des classes de formation. Dans ce contexte, la classification devient un ensemble ouvert d'étiquettes où la présence d'observations de classes invisibles est possible.

Dans les applications où l'utilisateur s'intéresse à l'identification de quelques classes d'un grand univers de classification, la méthode la plus classique consiste à fusionner l'ensemble de classes non intéressantes en un seul grand ensemble négatif, ce qui rend généralement l'ensemble de données extrêmement déséquilibré. Dans ce cas, le classificateur est submergé par des observations négatives qui em-

pêchent la discrimination des classes positives. Certaines tentatives sont apparues pour tenter de remédier à cette situation, reposant principalement sur l'échantillonnage d'un sous-ensemble de représentants des négatifs (Aggarwal, 2013).

Cependant, il est très difficile et en quelque sorte injuste de réduire tous les points négatifs à un petit résumé qui peut ne pas être suffisant pour représenter toutes les possibilités. Une transformation plus appropriée de ce problème est une classification ouverte, dans laquelle seules les classes positives sont modélisées lors de la formation et toute observation qui s'écarte remarquablement de la distribution des classes connues est rejetée.

Très peu de travaux ont abordé la classification dans un espace ouvert dans la littérature. Scheirer *et al.* (2013) ont présenté une formalisation de la classification ouverte et ont montré son importance dans les applications du monde réel. Les auteurs ont discuté du biais lié à l'évaluation des approches d'apprentissage et de la manière dont les précisions de reconnaissance sont gonflées dans des scénarios fermés, conduisant à une confiance surestimée dans les approches évaluées (Torralba et Efros, 2011). Dans la classification des ensembles fermés binaires, SVM définit un hyperplan qui sépare au mieux les deux classes. Scheirer *et al.* (2013) ont proposé un classificateur multiclasses ouvert basé sur SVM qui définit un hyperplan supplémentaire pour chaque classe, de sorte que celle-ci soit délimitée par deux hyperplans de l'espace des fonctions. Une instance de test est ensuite classée dans une classe de formation ou dans une classe inconnue en fonction de sa projection dans l'espace descriptif. Cette stratégie délimite chaque classe de formation de deux côtés. Mais, la "zone d'acceptation" de la classe est laissée illimitée dans la région située entre les hyperplans et aucun séparateur supplémentaire n'est fourni pour éviter la classification erronée d'instances inconnues appartenant à la classe. Notamment, c'est le cas des hyperplans liés mais éloignés.

Une autre approche d'apprentissage importante pour les problèmes ouverts est la classification à une classe. La technique la plus connue est la SVM à classe unique (Tax, 2001) dans laquelle le classificateur est formé uniquement sur une classe positive. Elle définit un contour qui le sépare du reste de l'univers de classification. Toute instance située en dehors de la limite de classe définie est considérée comme négative.

La classification à une classe est principalement utilisée dans la détection des valeurs aberrantes et des nouveautés. Elle est limitée à une classification de classe unique et ne peut pas être utilisée directement dans une classification à plusieurs classes. Un contre un et un contre tous (Rocha et Klein Goldenstein, 2014) sont des techniques populaires de classification multiclasses. Un contre un construit un modèle pour chaque paire de classes. Ensuite, les exemples de test sont évalués par rapport à tous les modèles construits. Un système de vote est appliqué et l'étiquette prévue est celle avec le plus grand nombre de voix. Un contre tous crée un classificateur unique par classe, avec les exemples de cette classe comme positifs et tous les autres exemples comme négatifs. Tous les classificateurs sont appliqués sur un exemple de test et l'étiquette prévue est celle avec le score le plus élevé. Il est possible d'envisager un classement avec la validation croisée comme méthode d'évaluation. De cette façon et de manière itérative, chaque classe sera utilisée en tant qu'entraînement positif et toutes les classes restantes (connues) en tant que reste de l'univers.

CHAPITRE II

ÉTAT DE L'ART

2.1 Hyperparamètres

Un hyperparamètre est un paramètre dont la valeur est définie avant le début du processus d'apprentissage, et qui concerne le processus d'entraînement lui-même. En revanche, les valeurs des autres paramètres sont dérivées et ajustées par l'entraînement sur des données existantes.

C'est une configuration externe au modèle et dont la valeur ne peut pas être estimée à partir des données. Ils sont souvent spécifiés par le praticien. Aussi, différents algorithmes d'apprentissage nécessitent différents hyperparamètres et il est souvent essentiel de définir correctement ces choix de paramètres pour réaliser le plein potentiel d'une méthode (Claesen et De Moor, 2015).

D'une manière générale, leurs valeurs sont décidées en définissant des valeurs différentes, en formant différents modèles et en déterminant ceux qui fonctionnent le mieux, en les testant.

Ces paramètres doivent souvent être réajustés lorsque l'algorithme est appliqué à un nouveau domaine de problèmes, et le processus de réglage lui-même dépend souvent de l'expérience et de l'intuition personnelle de l'expert d'une manière difficile à quantifier ou à décrire.

Par exemple, il a été démontré qu'il est possible d'améliorer l'état de la technique de référence dans la classification des images en changeant la configuration des méthodes existantes plutôt que d'inventer de nouveaux algorithmes d'apprentissages (Pinto *et al.*, 2009, Coates *et al.*, 2011)(Bergstra *et al.*, 2011, Snoek *et al.*, 2012, Thornton *et al.*, 2012). Il est parfois difficile de savoir si une méthode donnée est réellement meilleure ou tout simplement mieux réglée (Hutter *et al.*, 2014).

2.2 Méthodes d'optimisation

En apprentissage automatique, le processus d'autoapprentissage d'un algorithme correspond généralement à deux boucles imbriquées (figure 2.1). La boucle externe itère sur les valeurs des hyperparamètres, alors que la boucle interne minimise une mesure d'erreur empirique.

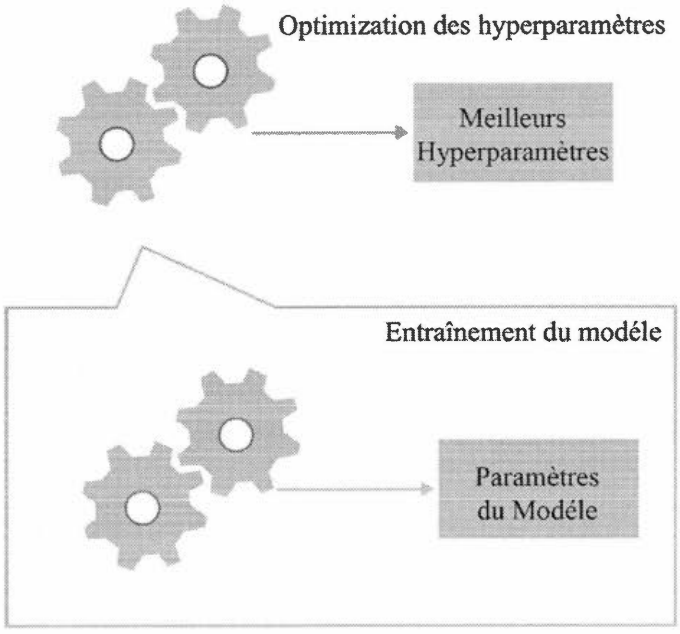


Figure 2.1: Processus d'autoapprentissage automatique
une configuration dans laquelle un algorithme d'optimisation maximise une fonction objective via une interface dite de boîte noire.

L'optimisation ou le réglage des hyperparamètres pose le problème de choisir un ensemble d'hyperparamètres optimaux pour un algorithme d'apprentissage donné. Ce domaine a bien progressé au cours de la dernière décennie, mais la performance de nombreuses méthodes d'apprentissage dépend toujours des fonctionnalités conçues manuellement ou du choix des hyperparamètres, ce qui fait souvent la différence entre des performances médiocres et d'autres plus optimales.

L'optimisation des hyperparamètres trouve un tuple de paramètres qui fournit un modèle optimal minimisant une fonction de perte prédéfinie sur des données indépendantes. La fonction objectif prend un tuple d'hyperparamètres et renvoie la perte associée (Hutter *et al.*, 2011).

Les travaux récents montrent que même les cadres d'apprentissage automatique très flexibles avec des centaines d'hyperparamètres peuvent être optimisés efficacement (Claesen et De Moor, 2015). Toutefois, si l'évaluation d'un seul hyperparamètre est déjà exigeante en calcul (comme c'est le cas, par exemple, pour de volumineuses séries de données), la réalisation d'une optimisation étendue sur tous les hyperparamètres, sur un grand espace ou un intervalle risque de coûter trop cher surtout que cette optimisation nécessite l'utilisation (et peut-être même l'abus) d'une technique de validation qui ressemble à une seconde phase d'apprentissage ou à une extension de l'algorithme d'apprentissage lui-même.

La métrique de performance (ou la fonction objectif) peut être visualisée sous forme de carte thermique dans l'espace de paramètre à n dimensions, ou sous forme de surface dans une dimension $(n + 1)$ espaces (la dimension $(n + 1)$ étant la valeur de cette fonction objectif). Plus cette surface est bosselée (plus les minima et points de selle sont locaux), plus il devient difficile d'optimiser ces paramètres.

Nous allons détailler les méthodes les plus populaires qui ont été suggérées pour

réaliser cette tâche d'optimisation.

2.2.1 Recherche Exhaustive

Pendant des décennies, la technique standard pour l'optimisation des hyperparamètres dans l'apprentissage automatique a été une simple recherche exhaustive à base de grille C'est une recherche à travers un sous ensemble spécifié de l'espace d'hyperparamètres d'un algorithme d'apprentissage (Bergstra et Bengio, 2012).

Cette recherche doit être guidée par une métrique de performance, généralement mesurée par une validation croisée sur l'ensemble d'entraînement ou d'une évaluation sur un ensemble de validation.

Comme l'espace paramétrique d'un algorithme d'apprentissage automatique peut inclure des valeurs réelles ou non réelles, liées ou non liées, la définition manuelle des limites et la définition des intervalles peuvent être nécessaires avant d'appliquer la recherche par grille. Par exemple, un classificateur SVM en marge souple, équipé d'un noyau linéaire, polynomial ou d'une fonction de base radiale est à ajuster pour de bonnes performances sur les données de vues : la régularisation de la constante C et de l'hyperparamètre de chaque type de noyau (figure 2.2).

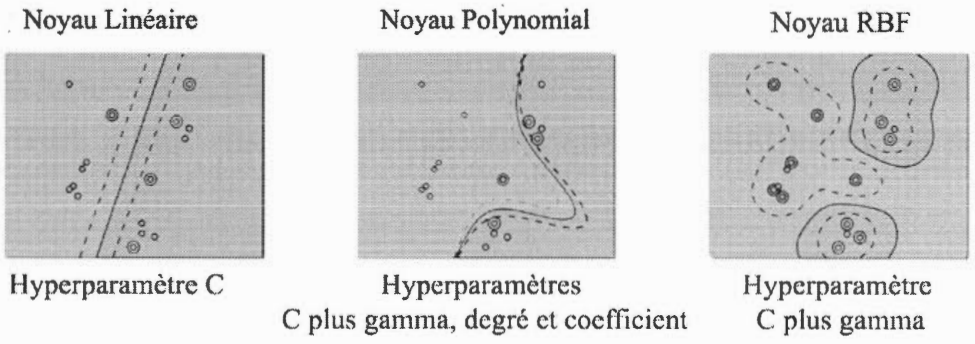


Figure 2.2: Noyaux SVM "standard" et exemple de limites résultantes

Chaque noyau est caractérisé par un ensemble d'hyperparamètres qui doivent être ajustés pour un problème particulier

Supposons qu'un modèle d'apprentissage automatique M prenne les hyperparamètres x, y et z. En recherche en grille, nous définissons d'abord la plage de valeurs pour chacun des hyperparamètres x, y et z. Nous pouvons considérer cela comme un tableau de valeurs pour chacun des hyperparamètres. Maintenant, la technique de recherche exhaustive construira de nombreuses versions de M avec toutes les combinaisons possibles de valeurs des hyperparamètres (x, y et z) que nous avons définies en premier lieu. Cette plage de valeurs des hyperparamètres est appelée une grille.

2.2.2 Algorithme de descente

En analyse numérique, l'algorithme de descente est une technique d'optimisation mathématique qui appartient à la famille de la recherche locale. C'est un algorithme itératif qui commence par une solution arbitraire à un problème, puis tente de trouver une meilleure solution en apportant un changement progressif à la solution. Si le changement produit une meilleure solution, un autre changement progressif est apporté à la nouvelle solution, et ainsi de suite jusqu'à ce qu'aucune autre amélioration ne puisse être trouvée (Cano *et al.*, 2005).

L'algorithme de descente trouve des solutions optimales aux problèmes convexes, par contre aux autres problèmes il ne trouvera que des optimums locaux (solutions qui ne peuvent être améliorées par aucun voisin), qui ne constituent pas nécessairement la meilleure solution possible (l'optimum global) parmi toutes les solutions envisageables (l'espace de recherche).

Cet algorithme est donc une méthode d'optimisation locale, il converge vers l'optimum local le plus proche mais le problème est qu'il n'arrive plus à s'en sortir. Il converge assez rapidement, mais pour obtenir une précision intéressante, d'autres améliorations sont nécessaires.

Pour éviter de rester bloqué dans les optimums locaux, nous pouvons utiliser des redémarrages (c.-à-d. recherche locale répétée), ou des schémas plus complexes basés sur des itérations (comme des itérations recherche locale), ou sur la mémoire (comme l'optimisation de la recherche réactive et la recherche taboue), ou sur des modifications stochastiques sans mémoire (comme un recuit simulé).

C'est un algorithme à tout moment : il peut renvoyer une solution valide même s'il est interrompu à tout moment, avant la fin. Pour une surface avec un seul maximum, l'algorithme de descente est bien adapté à l'optimisation et convergera vers le maximum global. Pour une surface avec deux maximums locaux (un seul d'entre eux est le maximum global), si l'algorithme de descente débute dans un mauvais emplacement, il peut converger vers le maximum le plus bas.

2.2.3 Optimisation bayésienne

L'optimisation bayésienne est une méthode d'optimisation globale pour les fonctions de boîte noire bruyantes. Elle vise à recueillir des observations révélant autant d'informations que possible sur cette fonction et, en particulier, sur l'emplacement de l'optimum. Elle tente d'équilibrer l'exploration (hyperparamètres dont le résultat est le plus incertain) et l'exploitation (hyperparamètres attendus proches de l'optimum).

Dans la pratique, il a été démontré que l'optimisation bayésienne obtenait de meilleurs résultats dans moins d'évaluations que la recherche par grille et la recherche aléatoire, en raison de la capacité de cette méthode de raisonner sur la qualité des expériences avant leurs exécutions. Cet algorithme utilise un modèle probabiliste M de la fonction f, basé sur des évaluations ponctuelles de f et sur tout état antérieur disponible. Le modèle M utilise les informations recueillies pour sélectionner les configurations suivantes à évaluer, et ensuite il met à jour M en se basant sur la nouvelle performance mesurée, et itère (Jones *et al.*, 1998,

Brochu *et al.*, 2010).

Les trois implémentations les plus populaires de l'optimisation bayésienne sont :

- Spearmint (Snoek *et al.*, 2012), qui utilise un processus gaussien (Rasmussen et Williams, 2005) modèle pour M ;
- SMAC (Hutter *et al.*, 2011), qui utilise des forêts aléatoires modifiées pour donner une estimation de l'incertitude (Hutter *et al.*, 2014) ;
- le Tree Parzen Estimator (Bergstra *et al.*, 2011), qui construit une densité estimée plus de bonnes et de mauvaises instanciations de chaque hyperparamètre pour construire M.

L'approche de Spearmint basée sur le processus gaussien est celle qui convient le mieux aux problèmes avec peu de calcul numérique (et pas d'autres). L'arborescence de SMAC et de TPE donnent les meilleurs résultats pour un espace hyperparamètrique de grande dimension et les problèmes d'optimisation discrets, notamment lors de l'optimisation des réseaux de neurones.

2.3 Galaxy-X et le raffinement

Galaxy-X, est une approche de classification multi-classes dans un espace ouvert. Pour chaque classe, Galaxy- X crée une hypersphère de délimitation minimale englobant toutes les instances pour chaque classe. De cette manière, il est capable de distinguer les instances nouvelles qui correspondent à la distribution d'une classe connue de celles qui en divergent (Dhifli et Diallo, 2016). Galaxy-X a un seul hyperparamètre : le raffinement pour l'ajustement du rayon des hypersphères englobantes. Le raffinement introduit une marge d'erreur au rayon de l'hypersphére. Il permet d'aggrandir le rayon et d'ajouter plus de généralisation ou de le rétrécir et dans ce cas permetttre plus de spécification aux modèles de classification (Dhifli et Diallo, 2016).

Il permet aussi la construction d'un classificateur simple multiclasses ouvert uti-

lisant une combinaison des concepts d'un classificateur à une seule classe et un classificateur multiclasses. Dans une première étape, toutes les classes de formation sont fusionnées en une seule grande "super-classe" et le classificateur d'une seule classe est formé sur l'ensemble super-classe. Dans ce cadre, le classificateur à classe unique rejettera et étiquettera comme inconnue toutes les instances de test qui ne correspondent pas à la distribution de toutes les classes de formation connues. Dans une deuxième étape, le classificateur multiclasses est formé sur les classes de formation d'origine et est utilisé pour classer les instances qui n'ont pas été rejetées par le classificateur à classe unique (Dhifli et Diallo, 2016).

Alors que l'approche théorique de la décision habituelle consiste à tenter d'estimer la meilleure règle de décision (limite) pour un ensemble particulier de distributions, cette méthode commence par l'observation fondamentale que toute limite de décision est parfaitement optimale pour de nombreuses distributions. Dans cette optique, Galaxy-X aborde le problème de classification d'une nouvelle manière dans laquelle nous sélectionnons certaines formes de frontières qui sont simples à mettre en œuvre et il démontre leur large applicabilité en décrivant les classes des distributions pour lesquelles ils sont la limite de partitionnement optimale.

L'hypersphère est la limite de division optimale pour une grande classe de probabilité paires de distribution (Cooper, 1962). La limite de décision de l'hypersphère, qui est implémentée simplement en comparant un seuil avec la distance euclidienne entre l'inconnu et un point fixe, peut être une excellente frontière d'approximation particulièrement dans des situations où les distributions de catégories affichent une symétrie sphérique et une différence dans la variance. C'est la limite de décision totalement optimale pour représenter le maximum de situations réelles possibles.

CHAPITRE III

PROBLÉMATIQUE ET MÉTHODOLOGIE

De nombreux algorithmes d'apprentissage automatique dépendent étroitement des choix des hyperparamètres. Ces choix peuvent avoir un impact énorme sur les performances du système. Par exemple, les travaux de Pinto *et al.* (2009) montrent que les performances les plus puissantes peuvent dépendre uniquement des choix d'hyperparamètres. La question « quelle est la qualité de ce modèle sur cet ensemble de données ? » est mal posée. Plutôt, il est plus logique de parler de la qualité de la meilleure configuration qui peut généralement être découverte par une procédure de recherche particulière dans un laps de temps donné.

Dans cette perspective, le réglage des hyperparamètres est un élément important pour comprendre la performance de l'algorithme, et devrait être formel et faire partie de l'évaluation du modèle.

La classification des séquences génomiques virales assigne une séquence donnée dans son groupe apparenté à de séquences connues. Les méthodes de classification existantes sont souvent conçues pour des familles bien étudiées de virus. Ainsi, les études génomiques comparatives virales pourraient bénéficier de solutions plus génériques et nécessitant moins de réglage.

Pour un jeu de données composé de séquences de virus, peut-on automatiquement

choisir les hyperparamètres optimaux tout en essayant de minimiser le budget de ressources informatiques telles que le temps d'exécution et / ou de computation et l'utilisation de la mémoire ?

Dans ce travail, nous proposons une approche d'automatisation des choix des hyperparamètres dans le but de fournir des outils pratiques qui remplacent le réglage manuel avec un processus d'optimisation reproductible et impartial et qui peuvent servir comme outils rapides pour la classification des souches nouvellement séquencées de diverses familles de virus. Notre approche consiste à formaliser les étapes de sélection des paramètres d'un modèle et évaluer ce modèle sur une tâche et des données virales.

Nous explorons la possibilité que l'optimisation manuelle ne soit plus suffisamment efficace pour justifier le manque de formalisation que cela implique.

Les développements récents dans la configuration des algorithmes d'apprentissage automatique permettent d'augmenter l'efficacité de la recherche des hyperparamètres, même dans des espaces de recherche mathématiquement maladroits, à un niveau où le résultat du réglage à la main peut être jumelé et dépassé en quelques heures sur un petit groupe d'ordinateurs.

La recherche automatique est surtout reproductible et prend donc en charge toutes les analyses incluant celles qui sont impossibles pour les chercheurs humains à optimiser. Avec un grand nombre d'entrées et une bonne fonction heuristique, nous essayons de trouver une bonne solution au problème même si cette solution peut ne pas être la solution optimale globale. Elle permet de gagner du temps et d'économiser les ressources.

Non seulement des hyperparamètres idéaux dictent la performance de l'entraînement mais ils contrôlent aussi la qualité des modèles prédictifs qui en résultent.

Si l'évaluation d'un seul hyperparamètre est déjà exigeante des calculs (comme c'est le cas, par exemple, pour les grands ensembles de données), effectuer une optimisation hyperparamétrique étendue sur un grand espace coûte trop cher. Trop réduire l'ensemble des hyperparamètres peut entraîner un algorithme inflexible et peut sacrifier une performance potentielle : capacité / flexibilité.

CHAPITRE IV

ARCHITECTURE ET ÉLÉMENTS CONCEPTUELS

4.1 Jeux de données

Nous avons testé note modèle sur plusieurs types de données, principalement des séquences de virus. Dans cette section, nous donnons une description complète de ces données et plus de détails peuvent être trouvés dans l'annexe A.

4.1.1 Jeux de données de référence

Nous avons d'abord testé nos algorithmes sur deux jeux de données de références :

- Iris : le jeu de données contient 3 classes de 50 instances chacune, chaque classe faisant référence à un type de fleurs d'iris.
- Digits : le jeu de données est composé de 1797 instances divisées en 10 classes représentant les chiffres arabes.

4.1.2 Séquences de virus représentées par la méthode RFLP

Nous avons choisi et utilisé cinq jeux de données de l'ensemble des données de Remita *et al.* (2017). Les autres ont simulé les résultats de la technique RFLP (le polymorphisme de longueur des fragments de restriction), en utilisant une liste de 172 enzymes de restriction de type II qui coupent les séquences d'ADN avec précision à chaque occurrence du site de reconnaissance et en extrayant leurs sites

de reconnaissance.

Ensuite, ils ont simulé la digestion de restriction des séquences d'ADN par calcul afin de représenter les séquences virales par des mesures calculées à partir de la distribution des fragments résultants de la digestion.

4.1.3 Séquences de virus représentées par la méthode K-mers

Nous avons utilisé cinq jeux de données des k-mers (sous-séquences nucléotidiques de longueur k) qui ont été aussi créés dans notre laboratoire pour des séquences virales (Lebatteux *et al.*, 2019).

On trouve une description plus détaillées des données dans l'annexe A.

4.2 Classification dans un espace ouvert avec Galaxy-X

Nous avons utilisé Galaxy-X (Dhifli et Diallo, 2016) comme exemple à optimiser. Galaxy-X est une méthode de classification dans un ensemble ouvert développée dans notre laboratoire.

4.2.1 Algorithme original de Galaxy-X

C'est un algorithme de classification multiclasses pour les problèmes dans un univers ouvert. Il est capable de distinguer des instances ressemblant à des classes déjà vues de celles qui sont des classes non vues auparavant.

Galaxy-X offre une grande flexibilité grâce à un hyperparamètre : le raffinement (figure 4.1).

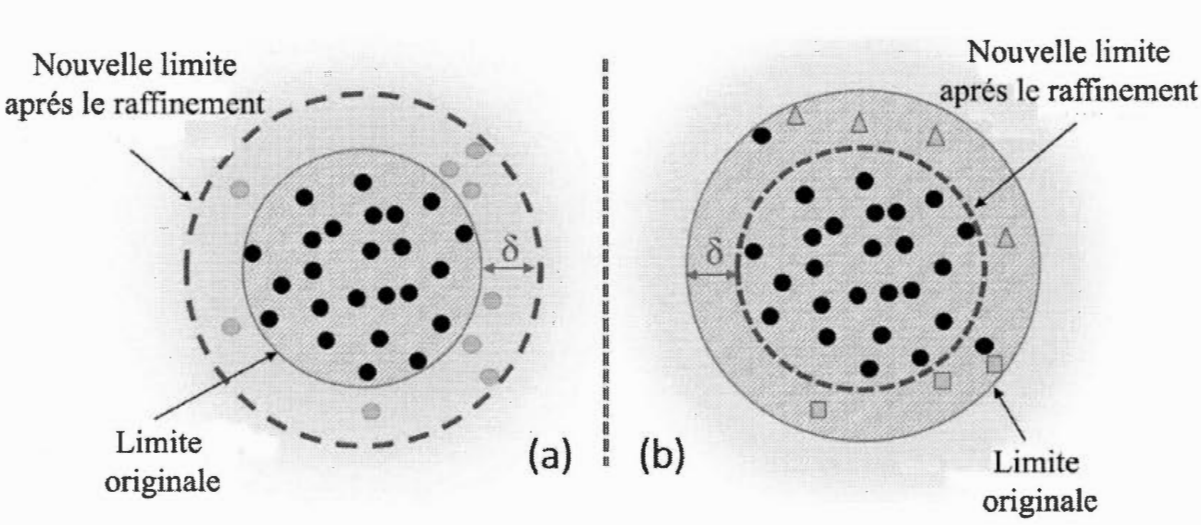


Figure 4.1: Le raffinement

Exemple de raffinement positif (a) et négatif (b) de la limite des classes. Les points noirs sont les instances des classes de l'entraînement. Les points gris sont des instances de test qui diffèrent légèrement de sa distribution (Dhifli et Diallo, 2016).

Cet hyperparamètre permet d'agrandir ou de réduire les limites de classe dépendamment de sa valeur. C'est un pourcentage qu'on applique au rayon de la sphère. S'il est positif, il va nous permettre de l'aggrandir et s'il est négatif de réduire sa taille ajoutant ainsi plus de généralisation ou de spécialisation aux modèles de classification (annexe B.1).

L'algorithme 1 décrit la phase d'apprentissage dans Galaxy-X. Étant donné un ensemble d'entraînement D et un ensemble de classes L défini sur D, un modèle M est créé pour chaque classe $l \in$ l'ensemble de classes L qui est composé de :

- La limite minimale de la classe ;
- Du centre qui est la moyenne des valeurs de cette classe ;
- Du rayon de l'hypersphère r qui est égal à la distance entre le centre et l'instance la plus divergente de l'ensemble d'entraînement.

Algorithme 1 : Galaxy-X : Le processus d'entraînement du modèle (Dhifli et Diallo, 2016)

Données : $\mathcal{D}$: Données d'entraînement, $\mathcal{L}$: Étiquettes d'entraînement

Résultat : $\mathcal{M}$: Ensemble des modèles

begin

  $\mathcal{M} \leftarrow \emptyset$

  foreach $l \in \mathcal{L}$ **do**

    $c_l \leftarrow Centrode(D_l)$

    $r_l \leftarrow Limite(D_l)$

    $\mathcal{M}_{\updownarrow} \leftarrow (c_l, r_l)$

    $\mathcal{M} \leftarrow \mathcal{M} \cup \mathcal{M}$

  end

end

4.2.2 Optimisations de l'algorithme Galaxy-X

Nous avons travaillé sur deux versions adaptées de Galaxy-X :

1. Galaxy-X -1 : Dans cette version, nous avons la même valeur de raffinement pour toutes les classes : Pour tenter d'estimer la meilleure règle de décision (limite) pour un ensemble particulier de distributions, Galaxy-X utilise un hyperparamètre : le raffinement qui est fixé manuellement par l'expert.

   Dans cette version éditée de Galaxy-X, nous utilisons deux différentes méthodes : la recherche exhaustive et l'algorithme de descente pour trouver la valeur optimale du raffinement qui donne les meilleurs résultats.

2. Galaxy-X -2 : Cette version nécessite une valeur de raffinement différente pour chaque classe :

   Galaxy-X fixe une valeur de raffinement qui change la limite des distribu-

tions pour toutes les classes. On a toujours la même valeur de raffinement pour toutes les classes. Pour améliorer cet algorithme, Nous avons défini une valeur de raffinement différente et unique pour chaque classe.

Algorithme 2 : Initialisation du tableau de raffinement

Entrées : y_train : Étiquettes d'entraînement

Sorties : tableau du raffinement initialisé à 0

begin

$lenSoft = longuer(y\_train)t$

for $i \in lenSoft$ **do**

Ajouter la valeur 0 *au tableau* ;

end

end

L'algotithme 2 explique comment les valeurs intiales de ces hyperparamètres sont introduits. Ensuite, nous avons utilisé différentes méthodes pour trouver et fixer les valeurs optimales et finales pour ces hyperparamètres :

- Recherche exhaustive
- L'algorithme de descente
- Recherche paramètre par paramètre, une contribution de ce mémoire et sera défini ultérieurement
- Optimisation Bayésienne

4.3 Algorithmes d'optimisation

Pendant des décennies, le standard *defacto* pour l'optimisation des hyperparamètres en apprentissage automatique a été une simple recherche exhaustive.

Dans certains cas, les experts ont appliqué une stratégie de raffinement manuel, consistant à utiliser une grille grossière pour ensuite s'adapter de manière itérative

à une région haute performance de l'espace hyperparamétrique.

Des méthodes d'optimisation bayésiennes séquentielles plus sophistiquées basées sur des modèles sont encore plus performantes et ont permis d'obtenir de nouveaux résultats à la pointe de la technologie pour plusieurs jeux de données (en optimisant bien plus de dimensions que ne le permettraient des méthodes simples, telles que la recherche exhaustive)

4.3.1 Recherche exhaustive

La recherche exhaustive ou recherche par force brute est une méthode algorithmique qui consiste principalement à essayer toutes les solutions.

Pour ce faire, nous générons d'abord la liste de toutes les combinaisons possibles pour les hyperparamètres et les enregistrons sous la forme d'une matrice. Après nous essayons chaque combinaison de la liste prédéfinie de valeurs et évaluons le modèle pour chaque combinaison (figure 4.2).

Une fois que toutes les combinaisons sont évaluées, le modèle avec l'ensemble de paramètres donnant la plus grande précision est considéré comme le meilleur et nous gardons ces valeurs comme hyperparamètres.

L'un des inconvénients majeurs de la recherche exhaustive est le fait que, lorsqu'il s'agit de dimensionnalité, elle souffre quand le nombre d'hyperparamètres augmente de manière exponentielle.

Avec seulement quatre paramètres, ce problème peut devenir impraticable, car le nombre d'évaluations requises pour cette stratégie augmente de manière exponentielle en le nombre de paramètres supplémentaires, en raison de la malédiction de la dimensionnalité. L'optimisation de l'hyperparamètre de Galaxy-X (le raffinement) avec une recherche exhaustive est décrite par l'algotithme 3.

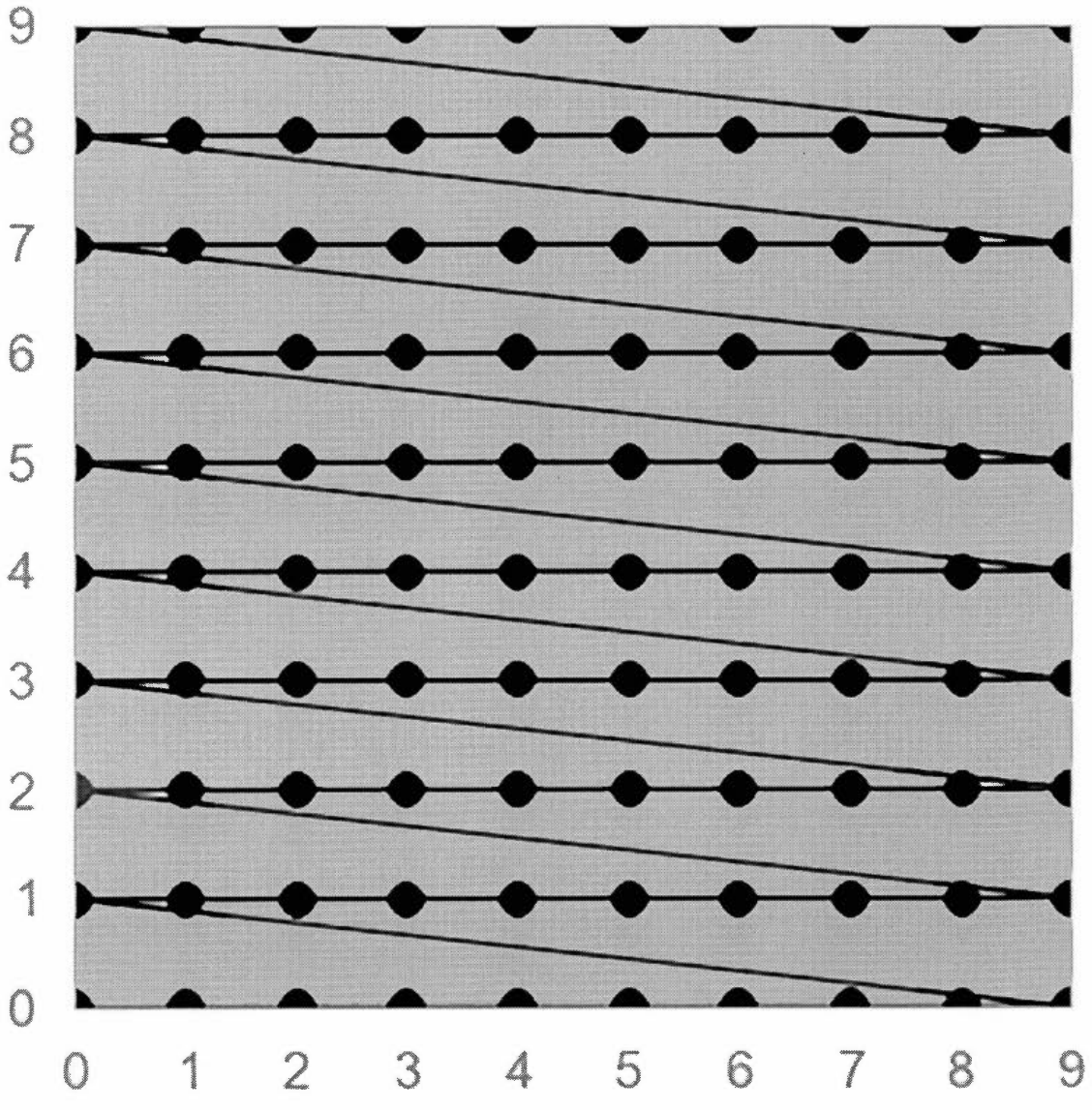


Figure 4.2: Recherche exhaustive

Deux hyperparamètres sont présentés sur les axes x et y pour essayer toutes les combinaisons possibles.

Algorithme 3 : Algorithme de recherche exhaustive

Entrées : tab : Liste des valeurs possibles, nombre : nombre de classes

Sorties : softToKeep : valeurs de raffinement optimisé, meilleurScore : score du meilleur classifieur

```
begin
    meilleurScore = 0
    combinaisons = Générer la liste de toutes les combinaisons possibles de tab
    for i in combinaisons do
        Exécuter Galaxy-X avec i comme valeur de raffinement
        Trouver le score de ces valeurs de i
        if score ≥ meilleurScore then
            softToKeep = i
            meilleurScore = score
        end
    end
end
```

4.3.2 Algorithme de descente

L'algorithme de descente est une technique d'optimisation mathématique qui appartient à la famille de la recherche locale. C'est un algorithme itératif qui commence par une solution arbitraire à un problème, puis tente de trouver une meilleure solution en apportant une modification incrémentale. Si le changement produit une meilleure solution, un autre changement progressif est apporté à la nouvelle solution, et ainsi de suite jusqu'à ce qu'aucune amélioration supplémentaire ne puisse être trouvée. L'algorithme d'optimisation de Galaxy-X avec cette approche est listé dans l'algorithme 4.

Algorithme 4 : Algorithme de descente

Entrées : intervalle : intervalle des valeurs possibles,

nb$_i$*trations* : *nombred'itrations*, *c* :

nombredeclasses **Sorties :** *softToKeep* : *valeursderaffinementoptimis*, *meilleurScore* : *sco*

```
    meilleurScore = 0
    combinaisons = Générer la liste de toutes les combinaisons possibles from
     tab
    comb = choisir un point de départ au hasard de l'intervalle
    for i in nbitrations do
        Générer comb1 et comb2 en respectant l'intervalle chacune dans une
         direction
        Exécuter Galaxy-X avec comb1 comme valeur de raffinement
        Trouver le score1
        Exécuter Galaxy-X avec comb2 comme valeur de raffinement
        Trouver le score2
        if score1 ≥ meilleurScore then
            softToKeep = comb1
            meilleurScore = score1
        end
        else if score2 ≥ meilleurScore then
            softToKeep = comb2
            meilleurScore = score2
        end
        else
            break
        end
    end
end
```

4.3.3 La recherche paramètre par paramètre

Dans notre cas, comme les hyperparamètres sont une marge d'erreur pour un rayon. Ils sont tous des valeurs numériques et compris entre -1.0 et 0.0. Cette méthode consiste à optimiser et à fixer la valeur d'un hyperparamètre avant de se concentrer sur le prochain pour éviter le problème de surdimensionnalité qu'on trouve avec une recherche exhaustive. L'optimisation de galaxy-X par cette nouvelle méthode est décrite dans l'algorithme 5.

Algorithme 5 : Algorithme de recherche paramètre par paramètre

Entrées : Combinaisons : liste de valeurs possibles, raffinement : tableau de raffinement

Sorties : softToKeep : valeurs de raffinement optimisé, meilleurScore : score du meilleur classifieur

```
begin
    meilleurScore = 0
    for index, valeur in raffinement do
        for x in combinaisons do
            raffinement[index] = x Exécuter Galaxy-X avec cette valeur de
             raffinement
            Trouver le score
            if score ≥ meilleurScore then
                softToKeep = raffinement
                meilleurScore = score
            end
        end
    end
end
```

4.3.4 Optimisation bayésienne

Nous avons utilisé une librairie d'optimisation globale construite sur l'inférence bayésienne et le processus gaussien, qui tente de trouver la valeur maximale d'une fonction inconnue en aussi peu d'itérations que possible. Cette technique est particulièrement adaptée à l'optimisation de fonctions coûteuses, situations dans lesquelles l'équilibre entre exploration et exploitation est important (Brochu *et al.*, 2010).

L'optimisation bayésienne fonctionne en construisant une distribution postérieure de fonctions (processus gaussien) qui décrit le mieux la fonction que nous souhaitons optimiser. À mesure que le nombre d'observations augmente, la distribution postérieure s'améliore et l'algorithme devient plus sûr des régions de l'espace des paramètres qui méritent d'être explorées et de celles qui ne le sont pas, comme le montre la figure 4.3.

Au fur et à mesure des itérations, l'algorithme équilibre ses besoins d'exploration et d'exploitation en tenant compte de ce qu'il sait sur la fonction cible. À chaque étape, un processus gaussien est ajusté aux échantillons connus (points précédemment explorés), et la distribution postérieure, associée à une stratégie d'exploration (telle que la limite de confiance supérieure ou l'amélioration attendue), est utilisée pour décider du prochain point qui devrait être exploré.

Ce processus est conçu pour minimiser le nombre d'étapes nécessaires pour trouver une combinaison de paramètres proche de la combinaison optimale. Pour ce faire, cette méthode utilise un problème d'optimisation de proxy (trouver le maximum de la fonction d'acquisition) qui, bien qu'il reste un problème difficile, est moins coûteux (au sens de calcul) et que des outils communs peuvent être utilisés (Algorithme 6). Par conséquent, l'optimisation bayésienne convient le mieux aux situations dans lesquelles l'échantillonnage de la fonction à optimiser est une tâche

très coûteuse (Snoek *et al.*, 2012).

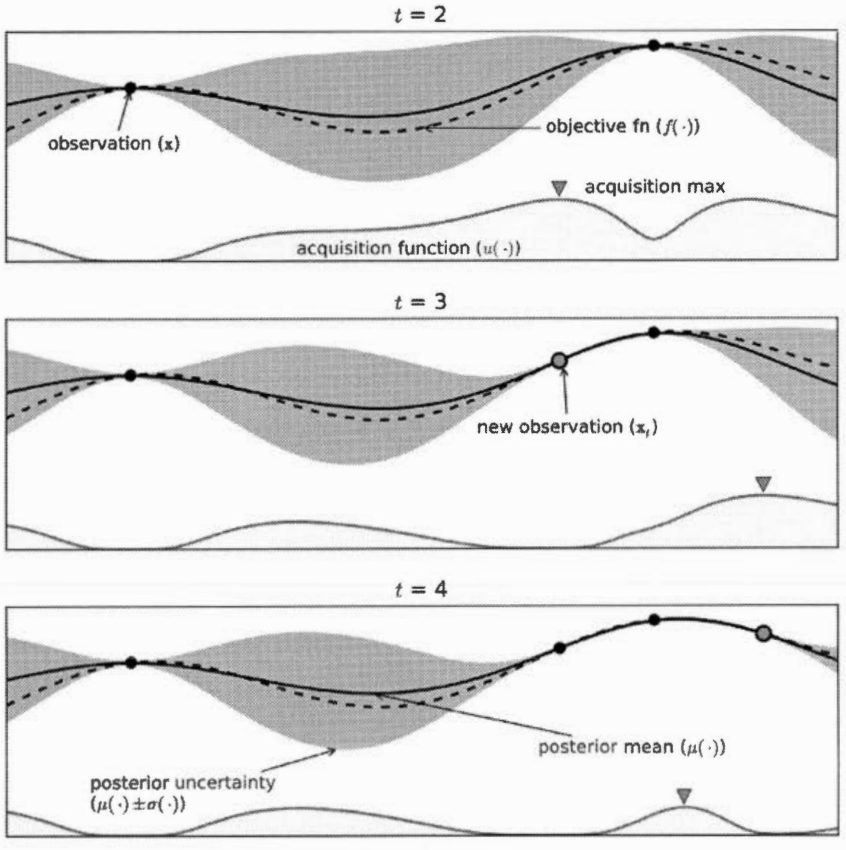


Figure 4.3: Exemple d'utilisation de l'optimisation bayésienne

Une approximation par processus Gaussien de la fonction objectif sur quatre itérations de valeurs échantillonnées de la fonction objectif. La figure montre également la fonction d'acquisition dans les parcelles ombrées inférieures (Brochu et al., *2010).*

Dans l'algorithme 9, nous présentons les différentes étapes d'optimisation bayésienne dans la bibliothèque que nous avons utilisée avec t égale au nombre d'itérations précisée à l'avance.

Algorithme 6 : Optimisation Bayésienne (Snoek *et al.*, 2012)

begin

  for $t = 1, 2..$ **do**

    Trouver x_t *en optimisant la fonction d'acquisition avec un Processus Gaussien*

    Echantillonner la fonction objectif

    augmenter les donnees et mettre a jour le Processus Gaussien

  end

end

4.4 Méthodes d'évaluation

En statistique et en apprentissage automatique, nous divisons généralement nos données en deux sous-ensembles : les données d'entraînement et les données de test. Nous adaptons notre modèle aux données d'entraînement, pour ensuite faire les prévisions sur les données de test.

4.4.1 Division en ensemble de test et ensemble d'entraînement

Cette méthode divise de façon aléatoire l'ensemble de données en un ensemble de test et un ensemble d'entraînement avec des tailles choisies au préalable (voir figure 4.4).

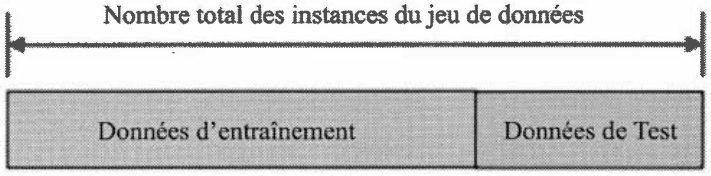


Figure 4.4: Exemple de division des données

Cette méthode divise l'ensemble de données avec un pourcentage prédéfini en données d'entraînement et données de test

Pour ce faire, nous avons utilisé la librairie python *sklearn* : *train_test_split* comme détaillée dans l'algorithme 7. Les jeux données seront divisées par conséquent en :

— 33% des données comme données de test ;

— 67% des données comme données d'entraînement.

Mais cette méthode n'est pas très fiable et peut causer un risque comme par exemple dans le cas où une classe de nos données sera exclu ou qu'une classe soit beaucoup mieux représentée que les autres, il en résultera un sur-ajustement des données. Le modèle sera très performants sur les données d'apprentissage, mais probablement très inexact sur de nouvelles instances.

Algorithme 7 : Division du jeu de données

Entrées : X : Données, y : Étiquettes

Sorties : X_train : Données d'entraînement, X_test : Données de test, y_train : Étiquettes d'entraînement, y_test : Étiquettes de test

begin

importer train_test_split

$X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.33)$

end

4.4.2 Validation croisée

La validation croisée est une bonne technique d'évaluation de la performance en généralisation d'un classificateur. L'idée derrière cette technique est de tester le modèle du classificateur obtenu sur des données qui n'ont pas participé à l'apprentissage, afin de permettre de prédire le comportement du classificateur face aux nouvelles données.

En pratique, on parle de validation croisée k-itérations qui consiste à diviser l'ensemble de validations en k sous-ensembles (figure 4.4). On utilise alors un sous-ensemble pour tester le modèle issu de l'apprentissage effectué avec les k-1 autres sous-ensembles. On répète k fois cette opération à travers tous les k sous-ensembles formés. La moyenne des erreurs déterminées lors des k opérations permet de donner une estimation de la capacité de généralisation du classificateur. La variance des résultats obtenus est réduite lorsque k croît, mais l'inconvénient de cette méthode est le temps de calcul qui devient très long.

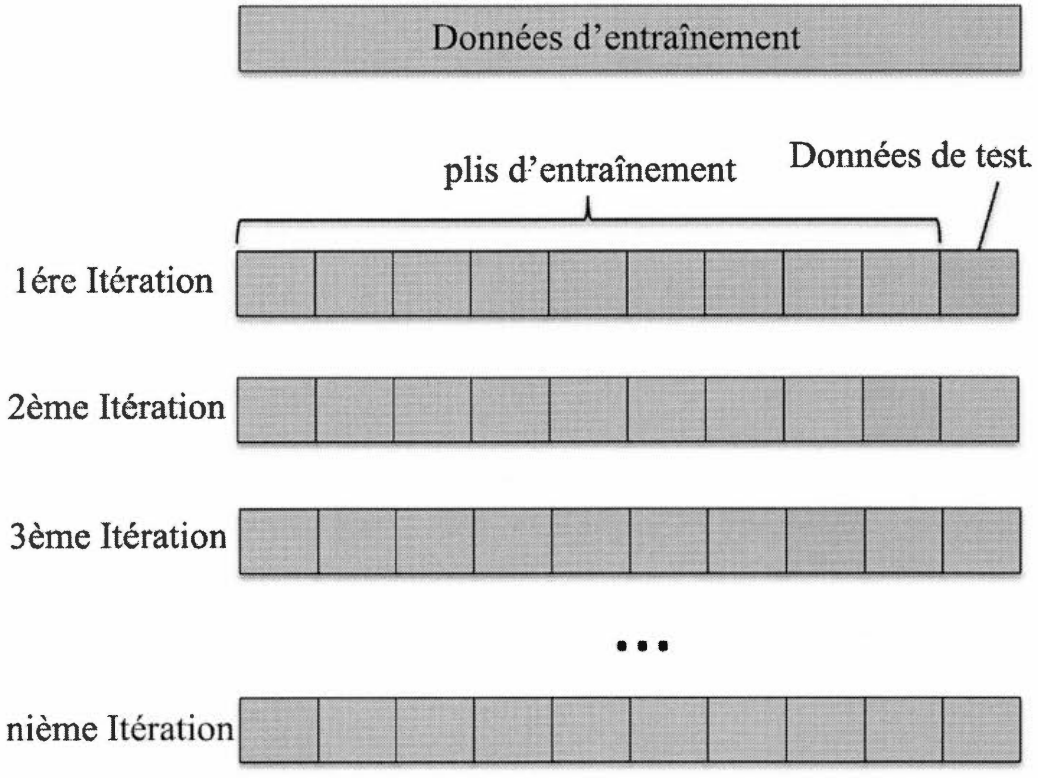


Figure 4.5: Validation croisée

Utilisation d'une technique d'échantillonage pour estimer fiablement les résultats d'un modèle

L'algorithme 4 explique comment nous avons procédé pour réaliser cette tâche et

appliquer la validation croisée à l'ensemble de nos données.

La méthode *kFold* partage le jeu de données selon la variable *n_splits* qui indique le nombre d'itérations ou de plis, égal à deux dans cet exemple. Elle fournit des indices pour fractionner les données en des ensembles d'entraînement et de test sans réarranger l'ordre fournit par défaut.

Algorithme 8 : La validation croisée

Entrées : X : Données, y : Étiquettes

Sorties : X_train : Données d'entraînement, X_test : Données de test, y_train : Étiquettes d'entraînement, y_test : Étiquettes de test

begin

importer kFold

$kf = KFold(n\_splits = 2)$

$kf.get\_n\_splits(X)$

for $X\_train, y\_train \in kf.split(X)$ **do**

$X\_train, X\_test = X[train\_index], X[test\_index]$

$y\_train, y\_test = y[train\_index], y[test\_index]$

end

end

4.4.3 Validation leave-p-out

Nous avons aussi ajouté la méthode *'leave_p_out'* de la validation croisée adaptée de (Dhifli et Diallo, 2016). Elle est détaillée dans l'algorithme 9.

Cet algorithme implique l'utilisation de p sous-ensembles comme jeu de test et les sous-ensembles restants comme jeu d'entraînement, avec p égal au nombre d'étiquettes à exclure dans chaque itération. Ceci est répété sur toutes les combinaisons possible de découpage du jeu de données original et se traduit par des tests sur

tous les échantillons distincts de taille p.

En raison du nombre élevé d'itérations qui croissent de manière combinatoire avec le nombre d'échantillons, cette méthode de validation croisée peut être très coûteuse. C'est la raison pour laquelle, nous devons toujours préciser α : le nombre maximal d'itérations et un compteur $\mathcal{N}$ du nombre d'itérations.

Algorithme 9 : La validation croisée avec 'leave_p_out' (Dhifli et Diallo, 2016)

Entrées : $\mathcal{D}$: Données d'entraînement, $\mathcal{L}$: Étiquettes d'entraînement, α : nombre maximal d'itérations, $\mathcal{P}$: nombre d'étiquettes à exclure dans chaque itération, $\mathcal{N}$: Nombre d'itérations, $\mathcal{E}$: le classificateur

Sorties : Scores : scores de la classification

begin

- $\mathcal{C} \leftarrow$ *toutes les combinaisons possibles de* $\mathcal{P}$ *de* $\mathcal{L}$
- **while** $(\alpha > 0)$ *et* $(\mathcal{C} \neq \emptyset)$ **do**
  - *Choisir au hasard une combinaison comb de* $\mathcal{C}$
  - $exclus \leftarrow Toutes\ les\ instance\ de\ D_{l_{comb}|\forall l_{comb}},\ comb \subseteq \mathcal{L}$
  - **foreach** $TrainingSet, TestSet \in \mathcal{N} - ValidationCroise(\mathcal{D} \setminus exclus)$ **do**
    - $Entrainer(\mathcal{E}, TrainingSet)$
    - $TestSet \leftarrow TestSet \cup exclus$
    - $PredictedLabels \leftarrow Prdire(\mathcal{E}, TestSet)$
    - $Scores \leftarrow Scores \cup Statistiques(PredictedLabels)$
  - **end**
  - $\mathcal{C} \leftarrow \mathcal{C} \setminus comb$
  - $\alpha \leftarrow \alpha - 1$
- **end**

end

4.5 Métriques d'évaluation

4.5.1 Rappel

Le rappel est le rapport :

$$F1 = \frac{VP}{VP + FN}$$

Où VP est le nombre de vrais positifs et FN le nombre de faux négatifs. Le rappel est intuitivement la capacité du classificateur à trouver tous les échantillons positifs. La meilleure valeur est 1 et la pire est 0.

4.5.2 Précision

La précision est le rapport :

$$F1 = \frac{VP}{VP + FP}$$

Où VP est le nombre de vrais positifs et FP le nombre de faux positifs. La précision est intuitivement la capacité du classificateur à ne pas étiqueter comme positif un échantillon négatif. La meilleure valeur est 1 et la pire est 0.

4.5.3 F-mesure

Le score F1, également appelé F score équilibré ou F mesure, est le rapport :

$$F1 = \frac{2 * precision * rappel}{precision + rappel}$$

Il peut être interprété comme une moyenne pondérée de la précision et du rappel, un score F1 atteignant sa meilleure valeur à 1 et le pire score à 0. Les contributions relatives de la précision et du rappel au score F1 sont égales.

CHAPITRE V

RÉSULTATS ET DISCUSSION

Généralement, les algorithmes d'apprentissage automatique ont des paramètres nécessaires à leur bon fonctionnement, appelés hyperparamètres, et qui permettent de réguler ou de modifier leur comportement à plus haut niveau. Habituellement, ces hyperparamètres sont choisis manuellement ou dans les meilleurs des cas par recherche exhaustive.

Des travaux récents ont souligné l'importance de l'utilisation de méthodes plus intelligentes pour l'optimisation des valeurs de ces hyperparamètres. En général, ce processus inclut :

1. Définir le modèle : Nous allons utiliser Galaxy-X (Dhifli et Diallo, 2016), outils de classification dans un univers ouvert.
2. Définir la plage de valeurs possibles pour tous les hyperparamètres : le raffinement est une valeur comprise entre –1 et 1.
3. Définir une méthode d'échantillonnage des valeurs des hyperparamètres (le but de ce projet). Plusieurs méthodes sont détaillées une à une et comparées dans ce qui suit.
4. Définir un critère d'évaluation pour juger le modèle : Pour avoir le maximum de détails, nous avons retenu le rappel, la précision et la F-mesure.
5. Définir une méthode de validation croisée : Leave-p-out.

Pour notre travail, nous avons appliqué plusieurs méthodes d'optimisation sur Galaxy-X, un outil de classification dans un univers ouvert.

Il est très important de pouvoir comparer différents algorithmes d'optimisation, afin de pouvoir choisir le plus performant pour un problème donné. Il en existe un très grand nombre, et nous nous sommes attachés à présenter les plus représentatifs et les plus courants.

Avant de présenter les méthodes d'optimisation que nous avons réalisées, nous avons apporté plusieurs changements à l'algorithme original de Galaxy-X (Dhifli et Diallo, 2016) et nous avons élaboré un outil de classification permettant de tester des méthodes d'optimisation sur plusieurs types de données. En plus de la version originale de Galaxy-X, nous avons ajouté deux nouvelles versions :

- Galaxy-X-1 : La première version où nous voulant optimiser la valeur du raffinement qui est le seul hyperparamètre de Galaxy-X avec deux méthodes :
  - algorithme de descente
  - recherche exhaustive
- Galaxy-X-2 : La deuxième version où le raffinement se présente comme un tableau où nous enregistrons une valeur pour chaque classe. Donc la taille du tableau est égale au nombre de classes et nous optimisons ce tableau avec quatre méthodes différentes :
  - algorithme de descente
  - recherche exhaustive
  - optimisation Bayésienne
  - une nouvelle méthode : recherche paramètre par paramètre.

Nous avons appliqué nos différentes méthodes sur plusieurs types de données :

- Jeux de données de référence : Afin d'avoir une base commune, mais éga-

lement pour avoir un ensemble représentatif d'une grande partie des problèmes que l'on peut retrouver dans la réalité, des protocoles de tests génériques ont été mis en place. Ces jeux de tests sont utilisés pour évaluer les performances et les capacités de convergence des algorithmes d'optimisation.

- Données virales réelles : permettant ainsi d'avoir un environnement d'étude hétérogène.
  - Séquences de virus représentées par la méthode K-Mers : Les jeux de données utilisés sont EBOSPECG pour le virus de Ebola, GEMGENCG pour les Geminivirus, HCVGENCG pour le virus de l'hépatite C, HIV-GRPCG pour le virus de l'immunodéficience humaine 1 (VIH-1), et enfin ROTSPECG pour le Rotavirus.
  - Séquences de virus représentées par la méthode RFLP : PMVHIVGC01 et PMVHIVPL01 pour le virus de l'immunodéficience humaine 1, PMV-HIVGC02, PMVHIVGC05 et PMVHIVPL02 pour le virus VIH-1 groupe M,

Le tableau A.1 (annexe A) offre des informations plus complète sur les jeux de données utilisés. Un point important est que la comparaison des performances d'algorithmes d'optimisation ne peut pas être mesurée par le nombre de problèmes qu'ils résolvent le mieux. Le théorème du *"no free lunch"* (Wolpert et Macready, 1997) montre qu'il ne peut pas exister une méthode résolvant tous les types de problèmes.

Ainsi, lorsqu'on compare deux algorithmes d'optimisation avec toutes les fonctions objectifs possibles, les performances des deux algorithmes seront semblables en moyenne.

Avec ce travail, nous prévoyons d'atteindre les objectifs suivants :

- Trouver les hyperparamètres optimaux, parmi un ensemble de solutions

candidates pour chaque jeu de données choisi.

— Choisir l'algorithme qui performe le mieux cet exercice.

5.1 La plateforme de classification

Nous avons développé une application en Python et Flask, conçue pour tester différentes méthodes d'optimisation sur un ensemble de données (figure 5.1).

Elle fournit plusieurs outils afin de permettre à l'utilisateur de visualiser ses données, de construire son modèle avec la méthode d'optimisation de son choix et de vérifier différentes mesures de performance.

À la fin de la simulation, l'outil produit des graphes, ainsi que des statistiques détaillées pour évaluer les performances des méthodes d'optimisation. L'utilisateur pourra ainsi comparer et décider de la méthode la plus adaptée à ses besoins.

Pour construire un modèle, l'utilisateur choisit son jeu de données à gauche, la méthode d'optimisation qu'il veut appliquer en haut, l'intervalle de valeur pour construire les combinaisons et l'ensemble de solutions possibles.

Ensuite il précise la version de Galaxy-X qu'il veut utiliser et finalement la méthode d'évaluation.

Cette partie optimise la fonction objectif pas à pas, car elle ne peut lancer qu'une itération de l'algorithme à chaque étape.

Nous avons décidé d'implémenter nos propres algorithmes pour ce type de problème sauf pour l'optimisation Bayésienne nous avons utilisé une libraire de Snoek *et al.* (2012).

Le but ultime de tout modèle d'apprentissage automatique est d'apprendre à partir d'exemples de manière à ce que le modèle soit capable de généraliser l'apprentis-

sage à de nouvelles instances qu'il n'a pas encore vues.

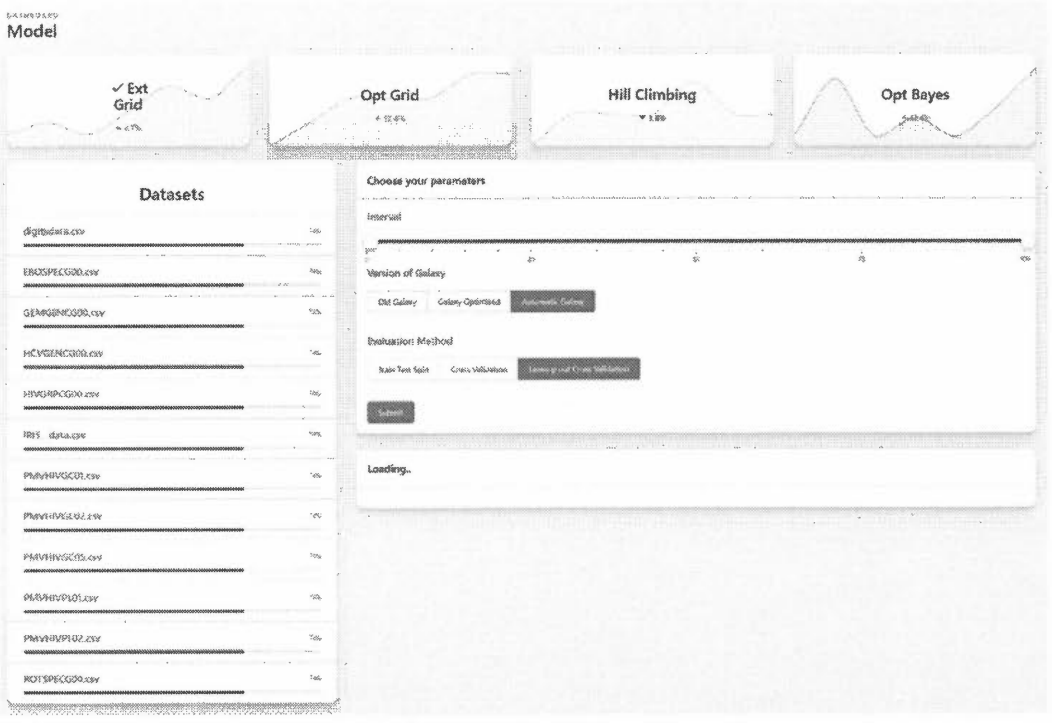


Figure 5.1: Capture d'écran de l'application de classification

Cette interface permet de construire un modèle pour chaque jeu de données avec différents paramètres et configurations

Nous devons entraîner le modèle sur un sous-ensemble de notre ensemble de données total, en conservant les données restantes pour évaluer la capacité du modèle à généraliser - autrement dit, nous devons être capables de répondre à la question "comment notre modèle réussira-t-il à traiter des données qu'il n'a pas apprises directement pendant l'entraînement ? "

Nous avons donc gardé tout au long de ce projet les résultats de deux méthodes d'évaluation :

— La division simple des données en données de test et données d'entraînement : *train_test_split* de *sklearn* ;

— La validation croisée : *leave_p_out* (Dhifli et Diallo, 2016)

5.2 Performance des différents algorithmes

5.2.1 Galaxy-X

En nous concentrant dans un premier temps sur les scores des métriques de performances obtenus pour les 12 ensembles de données avec l'algorithme original de Galaxy-X. Nous pouvons observer qu'il a une précision moyenne de 0.458, une F-mesure pondérée moyenne de 0,505 et un rappel moyen de 0.454 si on utilise une division en ensemble de test et ensemble d'entraînement comme méthode d'évaluation. Il a une précision moyenne de 0.821, une F-mesure pondérée moyenne de 0,790 et un rappel moyen de 0.818 si on utilise la validation croisée comme méthode d'évaluation.

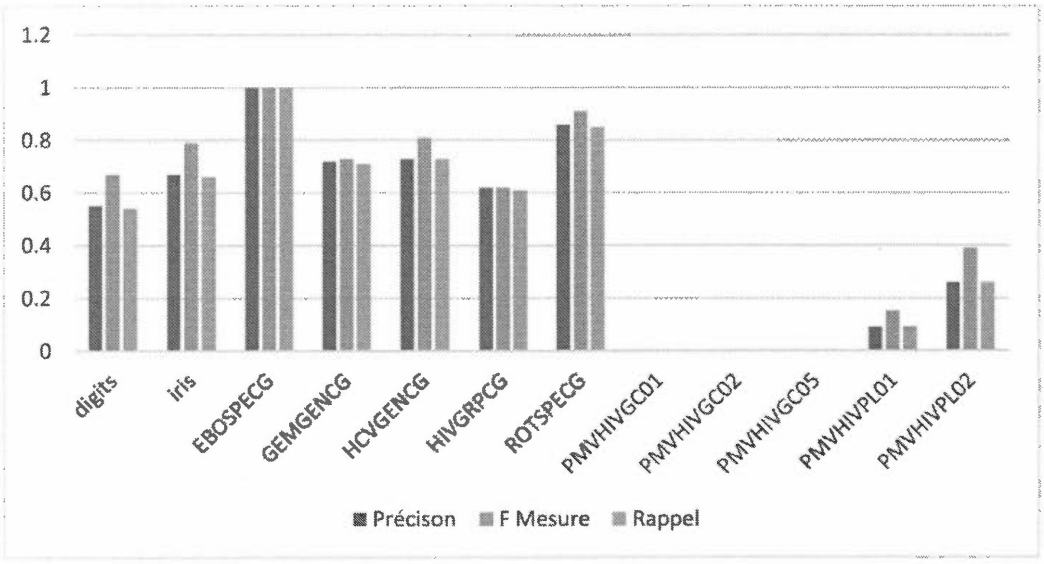
Nous remarquons que ces valeurs varient entre 0 et 1. Cet algorithme fonctionne très bien sur quelques données et très mal sur d'autres. Il se révèle plus performant avec 'Leave-p-out' parce que nous avons moins de classes (figure 5.2).

5.2.2 Galaxy-X -1 avec recherche exhaustive

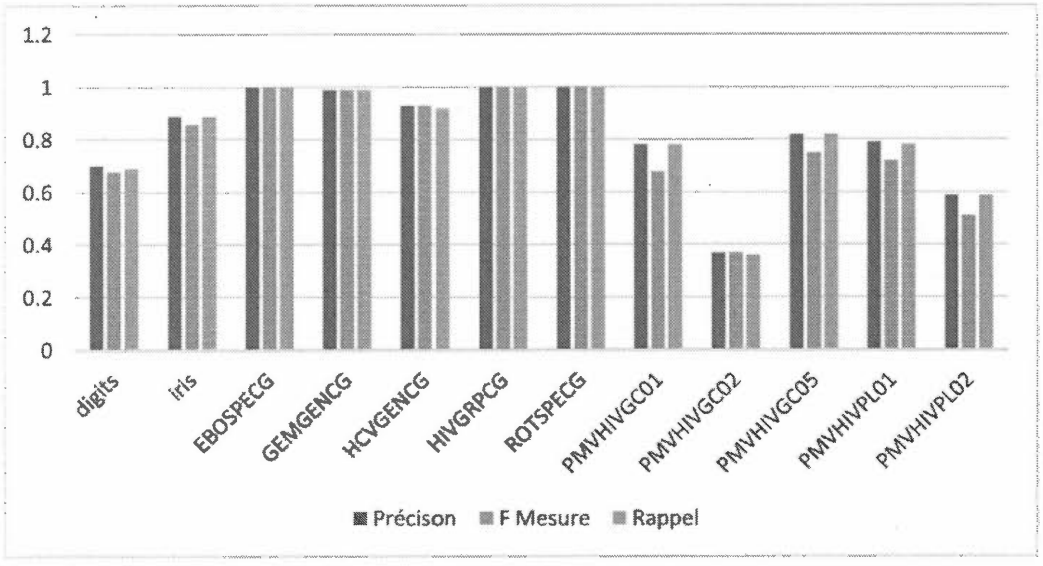
Nous avons ajouté un module à l'algorithme de Galaxy-X qui permet de faire une recherche exhaustive de tout l'ensemble de solutions possibles pour la valeur de raffinement et qui choisit la meilleure (pour plus de détails voir annexe B.3).

Notons que dans ce cas, comme nous avons une seule valeur à optimiser, cet ensemble de solutions est assez restreint. Il s'agit de l'intervalle [-1,1] donc les valeurs (-1, -0.9, -0.8, -0.7, -0.6, -0.5, -0.4, -0.3, -0.2, -0.1, 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1)

Cet algorithme construit simplement un modèle pour chaque solution possible, évalue chaque modèle et sélectionne l'architecture qui produit les meilleurs résul-



(a) Méthode de validation : Division simple

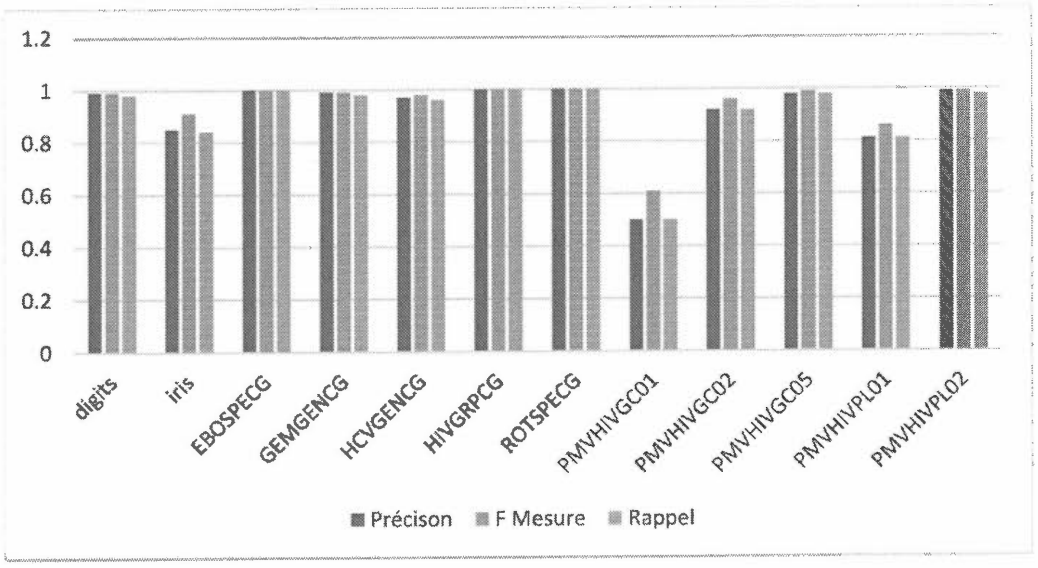


(b) Méthode de validation : Validation croisée

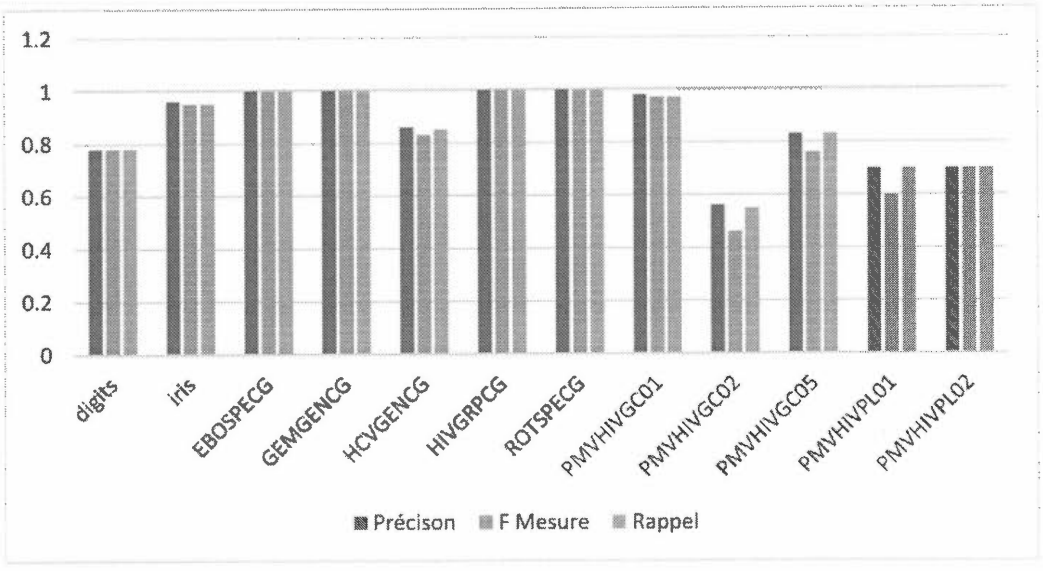
Figure 5.2: Performance de l'algorithme Galaxy-X

La figure illustre la performance de l'algorithme Galaxy-X sur des jeux de données viraux et les valeurs de précision, F Mesure et rappel avec deux méthodes d'évaluation

tats.



(a) Méthode de validation : Division simple



(b) Méthode de validation : Validation croisée

Figure 5.3: Performance de l'algorithme Galaxy-X -1 : Recherche exhaustive

La figure illustre la performance de l'algorithme Galaxy-X -1 : Recherche exhaustive sur des jeux de données viraux et les valeurs de précision, F Mesure et rappel avec deux méthodes d'évaluation

La figure 5.3 présente les scores des métriques de performances obtenus pour les 12 ensembles de données. Nous pouvons voir qu'il a une précision moyenne de 0.916, une F-mesure pondérée moyenne de 0,940 et un rappel moyen de 0.912 si on utilise une division en ensemble de test et ensemble d'entraînement comme méthode d'évaluation. Il a une précision moyenne de 0.864, une F-mesure pondérée moyenne de 0,837 et un rappel moyen de 0.860 si on utilise la validation croisée comme méthode d'évaluation.

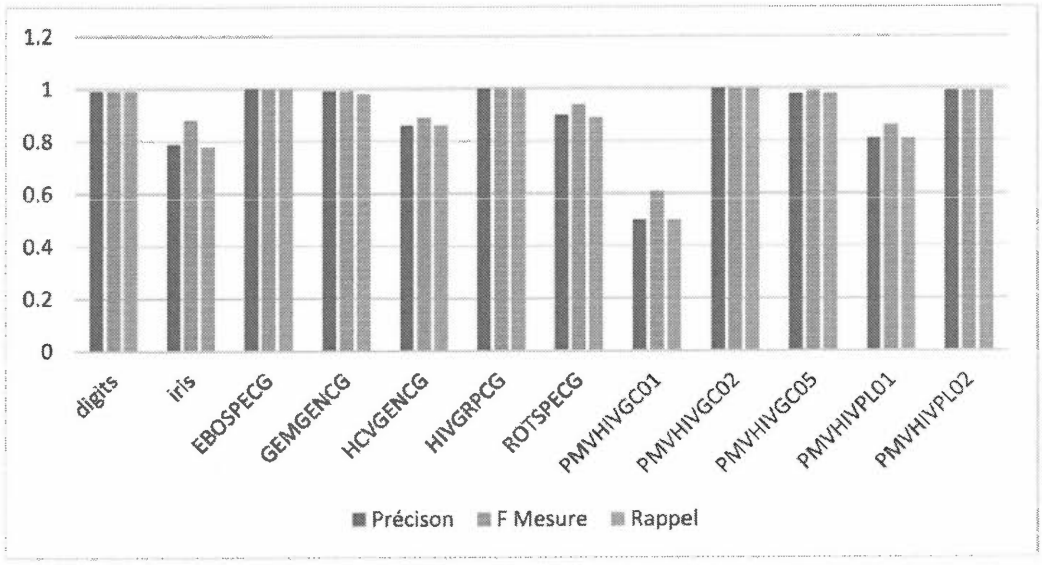
La recherche exhaustive améliore les scores de ces métriques en moyenne de 24% et les valeurs de l'hyperparamètre pour cet algorithme varient entre 0.5 et 1. C'est une méthode très performante qui permet d'améliorer assez rapidement la qualité du modèle surtout que dans ce cas nous n'avons qu'un seul hyperparamètre à optimiser. Cependant, il s'agit d'un échantillonnage exhaustif de l'espace hyperparamètre, il peut devenir assez inefficace si cet espace devient trop grand (figure 5.3).

5.2.3 Galaxy-X -1 avec L'algorithme de descente

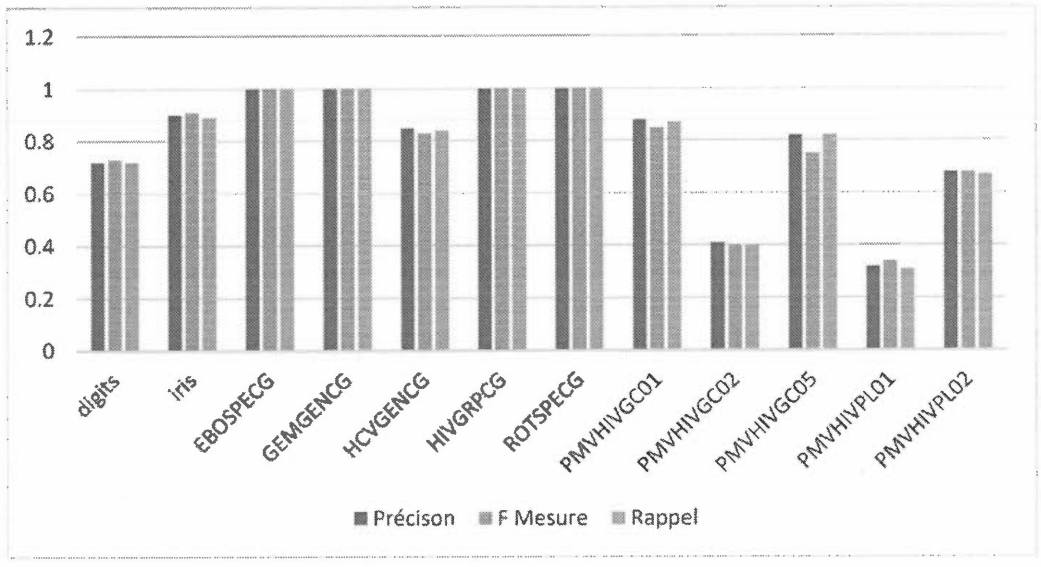
Nous avons implémenté un module qui ajoute la méthode de l'algorithme de descente pour optimiser la valeur du raffinement. Cette méthode vise à trouver la valeur qui donne la meilleure performance dans l'intervalle [-1, 1].

Notre méthode choisit une valeur au hasard de l'ensemble des valeurs et ensuite l'incrémente ou de la décrémente jusqu'à ce qu'elle trouve un sommet (pour plus de détails voir annexe B.4).

Pour ce projet et pour avoir des résultats reproductibles, nous avons forcé l'algorithme de toujours commencer au point (-0.3).



(a) Méthode de validation : Division simple



(b) Méthode de validation : Validation croisée

Figure 5.4: Performance de l'algorithme Galaxy-X -1 : L'algorithme de descente

La figure illustre la performance de l'algorithme Galaxy-X -1 : Algorithme de descente sur des jeux de données viraux et les valeurs de précision, F Mesure et rappel avec deux méthodes d'évaluation

La figure 5.4 présente les scores des métriques de performances obtenus pour les 12

ensembles de données Nous pouvons voir qu'il a une précision moyenne de 0.900, une F-mesure pondérée moyenne de 0,928 et un rappel moyen de 0.898 si on utilise la division simple comme méthode d'évaluation et une précision moyenne de 0.798, une F-mesure pondérée moyenne de 0,790 et un rappel moyen de 0.793 si on utilise la validation croisée comme méthode d'évaluation.

L'algorithme de descente améliore les scores de ces métriques en moyenne de 10% et les valeurs pour cet algorithme varient entre 0.32 et 1.

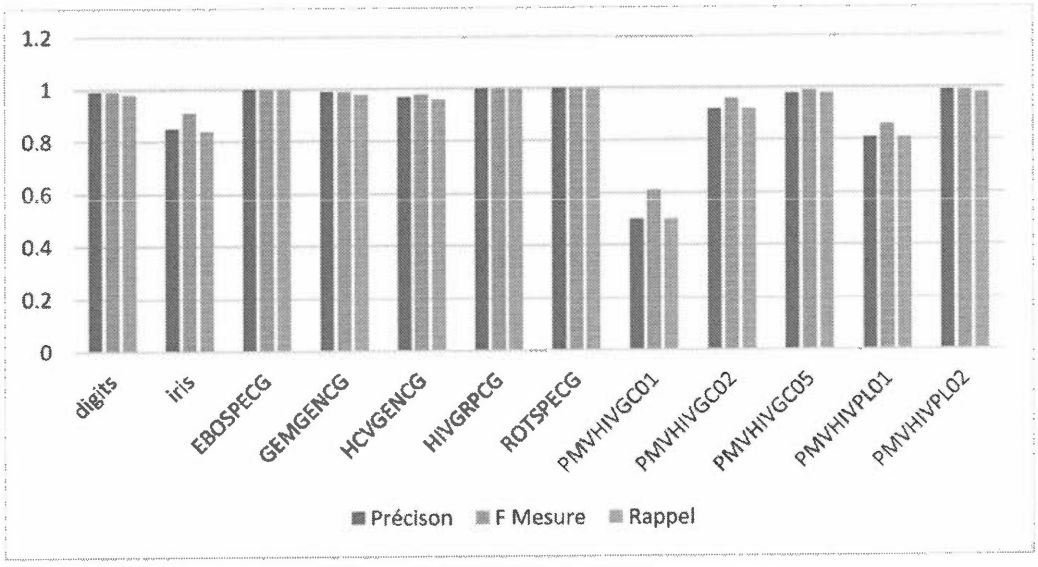
Nous concluons donc que cet algorithme dépend du point de lancement de l'Algorithme, il trouve toujours une solution performante mais pas la solution la plus performante. Les résultats dépendent du point de lancement de l'algorithme (figure 5.4).

5.2.4 Galaxy-X -2 avec Recherche exhaustive

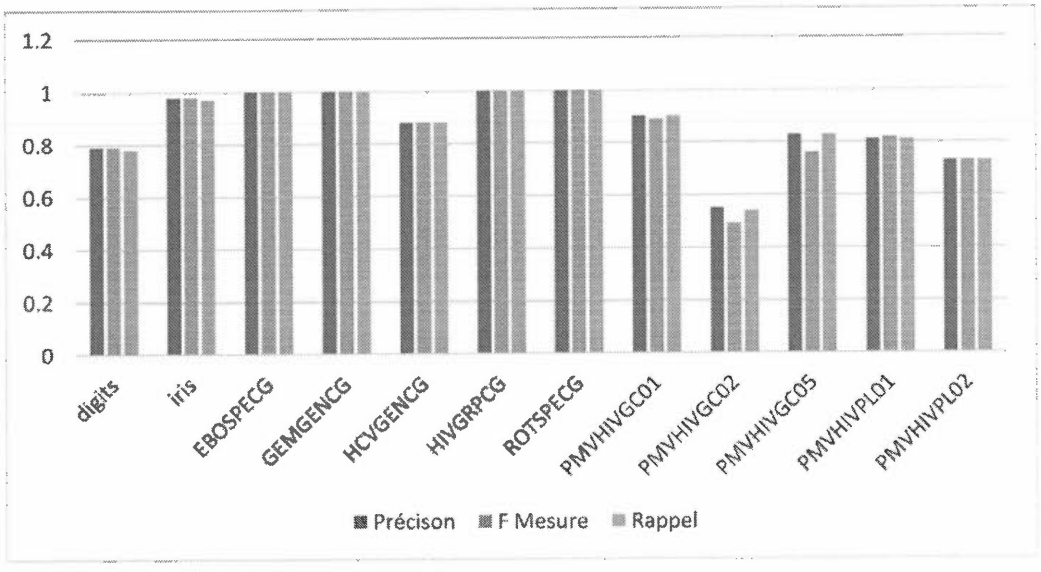
Nous avons changé le code de Galaxy-X pour rendre le raffinement un tableau qui contient une valeur unique pour chaque classe donc nous avons assez de valeurs dans le tableau que de nombre de classes.

Cet algorithme fait une recherche exhaustive de tout l'ensemble de solutions possibles pour la valeur optimale du raffinement et choisit la meilleure (pour plus de détails voir annexe B.5).

Notons que dans ce cas, comme nous avons plusieurs valeurs à optimiser, cet ensemble de solutions est assez grand de taille. Il s'agit de l'intervalle [-1,1] donc les valeurs (-1, -0.9, -0.8, -0.7, -0.6, -0.5, -0.4, -0.3, -0.2, -0.1, 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1) pour chaque valeur du tableau et toutes les combinaisons possibles entre eux.



(a) Méthode de validation : Division simple



(b) Méthode de validation : Validation croisée

Figure 5.5: Performance de l'algorithme Galaxy-X -2 : Recherche exhaustive

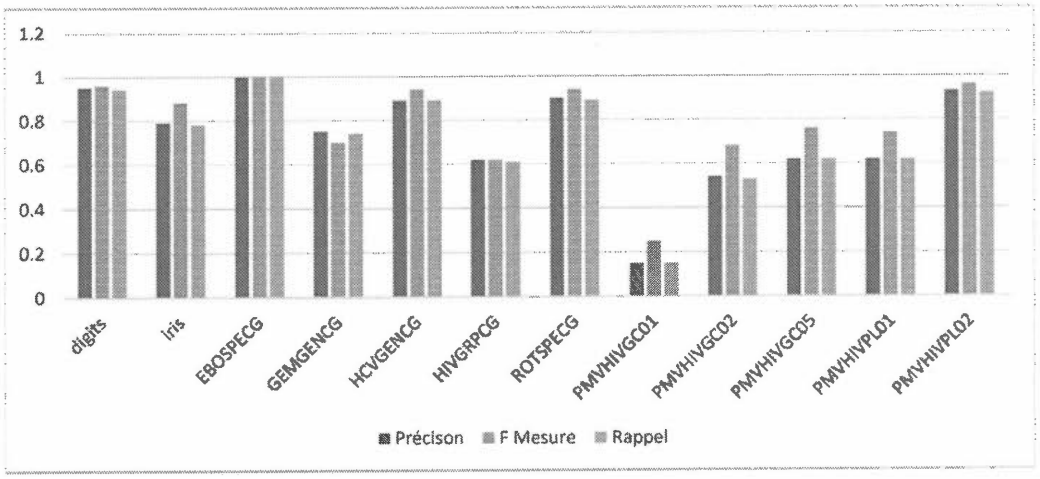
La figure illustre la performance de l'algorithme Galaxy-X -2 : Recherche exhaustive sur des jeux de données viraux et les valeurs de précision, F Mesure et rappel avec deux méthodes d'évaluation

La figure 5.5 présente les scores des métriques de performances obtenus pour les

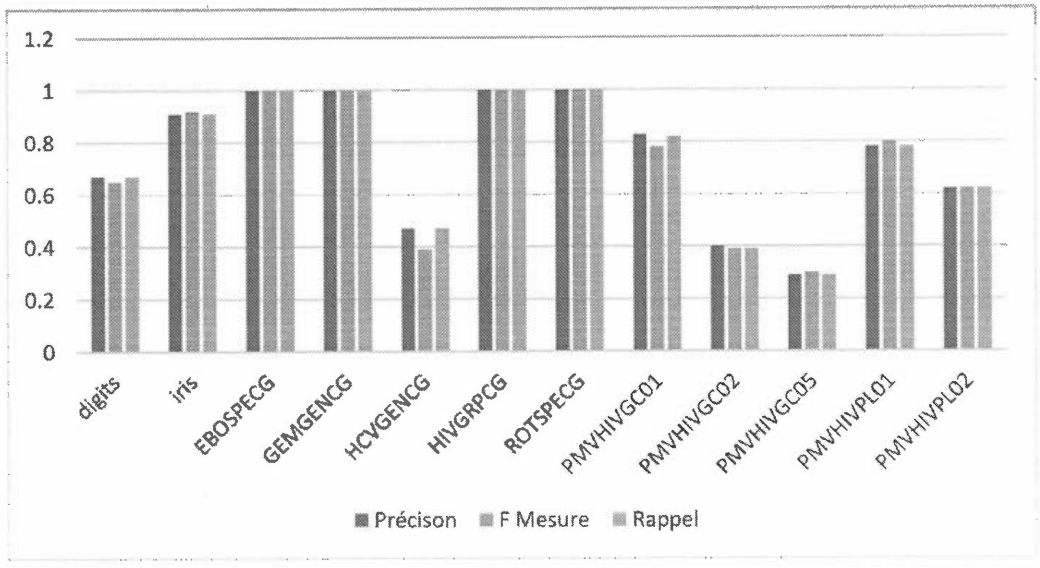
12 ensembles de données Nous pouvons voir qu'il a une précision moyenne de 0.916, une F-mesure pondérée moyenne de 0,940 et un rappel moyen de 0.912 si on utilise une division en ensemble de test et ensemble d'entraînement comme méthode d'évaluation. Il a une précision moyenne de 0.872, une F-mesure pondérée moyenne de 0,861 et un rappel moyen de 0.870 si on utilise la validation croisée comme méthode d'évaluation.

La recherche exhaustive améliore les scores de ces métriques en moyenne de 25% et les valeurs pour cet algorithme varient entre 0.5 et 1 (figure 5.5). Comme démontré dans la littérature, la recherche exhaustive donne les meilleurs résultats et garantie la meilleure performance d'un algorithme mais c'est un algorithme très gourmand en ressources puisqu'il teste toute les combinaisons possibles et ce nombre augmente exponentiellement avec le nombre de classes.

5.2.5 Galaxy-X-2 avec Algorithme de descente



(a) Méthode de validation : Division simple



(b) Méthode de validation : Validation croisée

Figure 5.6: Performance de l'algorithme Galaxy-X -2 : Algorithme de descente

La figure illustre la performance de l'algorithme Galaxy-X -2 : algorithme de descente sur des jeux de données viraux et les valeurs de précision, F Mesure et rappel avec deux méthodes d'évaluation

La figure 5.6 présente les scores des métriques de performances obtenus pour les 12 ensembles de données, nous pouvons voir qu'il a une précision moyenne de 0.730, une F-mesure pondérée moyenne de 0,785 et un rappel moyen de 0.724 si on utilise une division en ensemble de test et ensemble d'entraînement comme méthode d'évaluation et une précision moyenne de 0.747, une F-mesure pondérée moyenne de 0,737 et un rappel moyen de 0.745 si on utilise la validation croisée comme méthode d'évaluation.

L'algorithme de descente améliore les scores de ces métriques en moyenne de 10% et les valeurs pour cet algorithme varient entre 0.15 et 1 donc il est légèrement plus mauvais que la version dans Galaxy-X-1. Les résultats et les améliorations dépendent beaucoup d'un jeu de donnée à l'autre

Une description détaillée de l'algorithme de descente est présente dans l'annexe B.6.

5.2.6 Galaxy-X -2 avec Optimisation Bayésienne

L'optimisation bayésienne appartient à une classe d'algorithmes d'optimisation basée sur un modèle séquentiel (SMBO) qui permet d'utiliser les résultats de notre itération précédente pour améliorer notre méthode d'échantillonnage de l'expérience suivante (pour plus de détails voir annexe B.8).

Dans un premier temps, nous définirons un modèle construit avec des hyperparamètres λ qui, après avoir été formés, sont notés v par une mesure d'évaluation.

Ensuite, nous utilisons les valeurs d'hyperparamètre précédemment évaluées pour calculer une prévision postérieure de l'espace d'hyperparamètre. Nous pouvons ensuite choisir les valeurs optimales de l'hyperparamètre en fonction de cette prévision postérieure comme notre prochain modèle candidat.

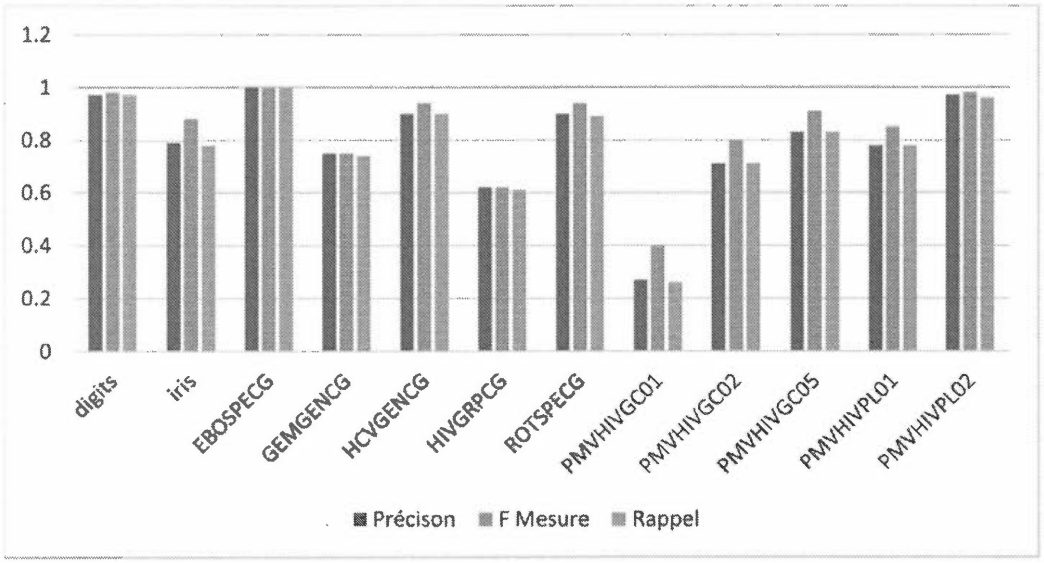
Nous répétons ce processus de manière itérative jusqu'à ce que la convergence soit optimale.

Nous utilisons un processus gaussien pour modéliser la probabilité antérieure des résultats dans l'espace des hyperparamètres. Ce modèle servira essentiellement à utiliser les valeurs des hyperparamètres $\lambda_1, ..., \lambda_i$ et les scores correspondants $v_1, ..., v_i$ observés pour approximer une fonction de score continue sur l'espace des hyperparamètres.

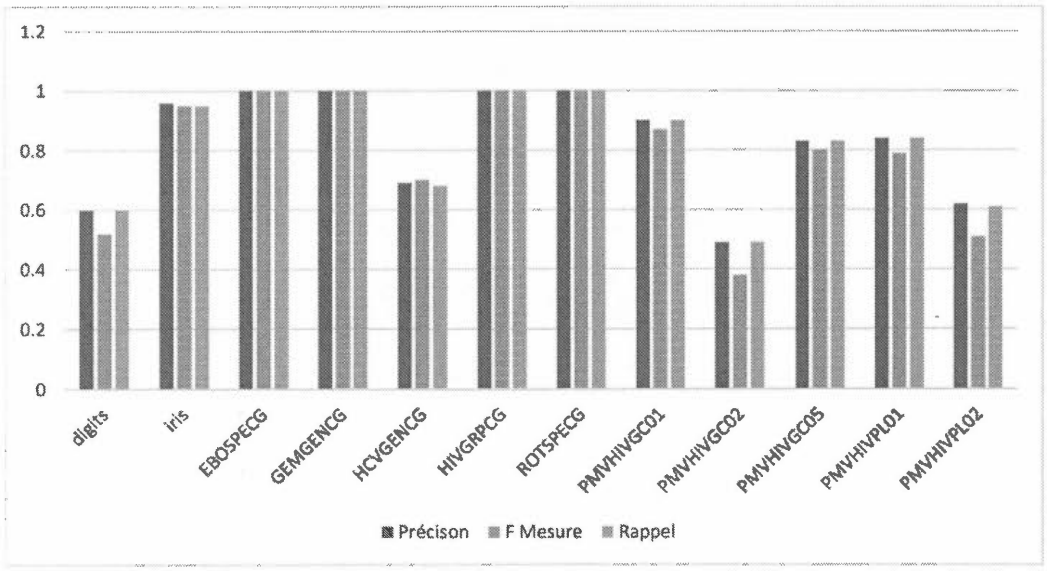
Cette fonction approximative inclut également le degré de certitude de notre estimation, que nous pouvons utiliser pour identifier les valeurs des hyperparamètres candidates qui produiraient l'amélioration attendue la plus importante par rapport au score actuel.

La formulation de l'amélioration attendue est connue sous le nom de notre fonction d'acquisition, qui représente la distribution postérieure de notre fonction de score dans l'espace hyperparamètrique.

La figure 5.7 présente les scores des métriques de performances obtenus pour les 12 ensembles de données. Nous pouvons voir qu'il a une précision moyenne de 0.790, une F-mesure pondérée moyenne de 0,837 et un rappel moyen de 0.785 si on utilise une division simple en ensemble de test et ensemble d'entraînement comme méthode d'évaluation. Il a une précision moyenne de 0.827, une F-mesure pondérée moyenne de 0,793 et un rappel moyen de 0.825 si on utilise la validation croisée comme méthode d'évaluation.



(a) Méthode de validation : Division simple

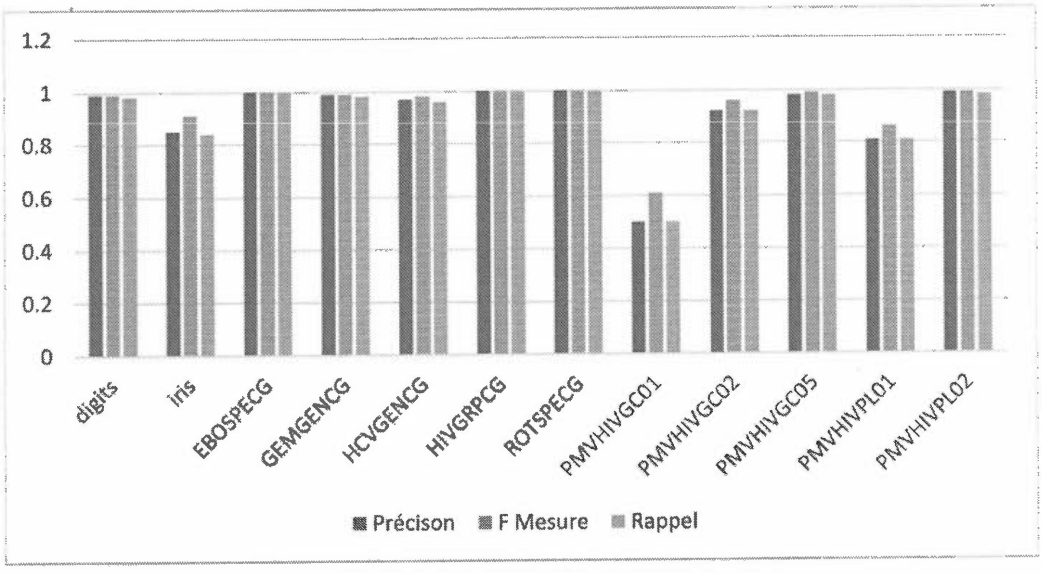


(b) Méthode de validation : Validation croisée

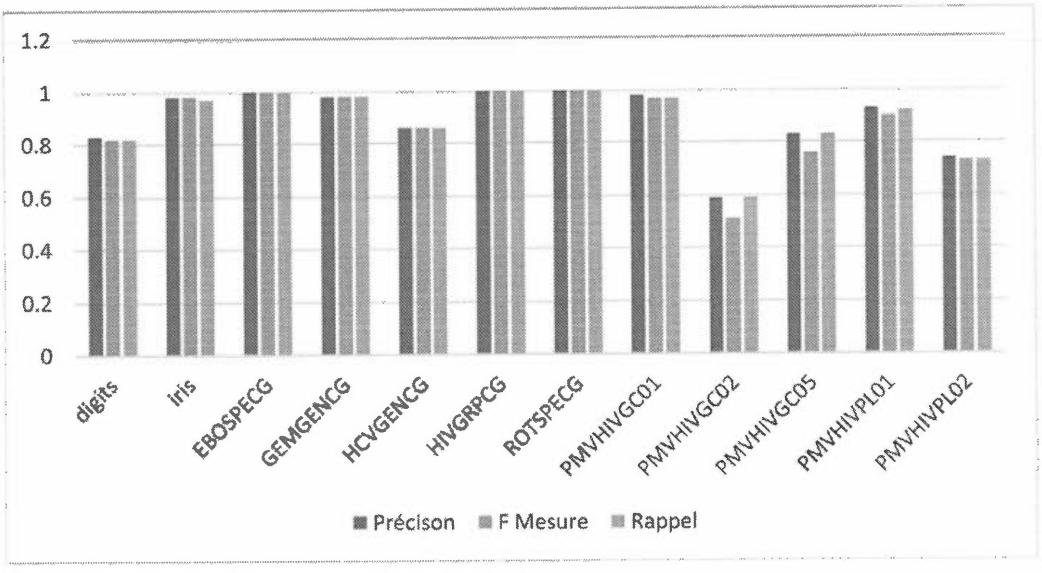
Figure 5.7: Performance de l'algorithme Galaxy-X -2 : Optimisation Bayésienne

La figure illustre la performance de l'algorithme Galaxy-X -2 :Optimisation bayésienne sur des jeux de données viraux et les valeurs de précision, F Mesure et rappel avec deux méthodes d'évaluation

5.2.7 Galaxy-X -2 avec Recherche paramètre par paramètre



(a) Méthode de validation : Division simple



(b) Méthode de validation : Validation croisée

Figure 5.8: Performance de l'algorithme Galaxy-X -2 : recherche paramètre par paramètre

La figure illustre la performance de l'algorithme Galaxy-X -2 : recherche paramètre par paramètre sur des jeux de données viraux et les valeurs de précision, F Mesure et rappel avec deux méthodes d'évaluation

Cette approche optimise chaque paramètre à la fois mais en tenant compte chaque fois des valeurs optimales trouvées précédemment (pour plus de détails voir annexe B.7).

Par exemple, si on a trois paramètres à optimiser x,y et z, l'algorithme va commencer par trouver la valeur optimale de x, ensuite il va optimiser y tout en tenant compte de cette valeur optimale de x qu'il a trouvé et finalement, il optimise z en sachant x et y.

La figure 5.8 présente les scores des métriques de performances obtenus pour les 12 ensembles de données. Nous pouvons voir qu'il a une précision moyenne de 0.916, une F-mesure pondérée moyenne de 0,940 et un rappel moyen de 0.912 si on utilise une division simple des données en ensemble de test et ensemble d'entraînement comme méthode d'évaluation. Il a une précision moyenne de 0.872, une F-mesure pondérée moyenne de 0,861 et un rappel moyen de 0.870 si on utilise la validation croisée comme méthode d'évaluation.

Tout comme la recherche exhaustive, cette méthode améliore les scores de ces métriques en moyenne de 25% et les valeurs pour cet algorithme varient entre 0.5 et 1 mais il teste beaucoup moins de combinaison donc consomme beaucoup moins de ressource pour atteindre ce même résultat.

5.2.8 Discussion et conclusion

En comparant la moyenne des résultats pour chaque algorithme (tableaux 5.1 et 5.2), nous remarquons que les algorithmes qui ont performé le mieux sont la recherche exhaustive et la recherche paramètre par paramètre.

L'algorithme de descente performe bien mais dépend du hasard lié au point de départ de l'algorithme. Quant à l'optimisation bayésienne, elle donne de bons résultats en raison de sa granularité, mais elle est imprédictible et consomme plus

de temps et de ressources quand il y a plus de classes.

Tableau 5.1: La moyenne des résultats pour chaque algorithme avec une division simple comme méthode d'évaluation

	Précision	F-Mesure	Rappel
Galaxy-X	0.458	0,505	0.454
Galaxy-X - 1 : Recherche exhaustive	0.916	0,940	0.912
Galaxy-X -1 : L'algorithme de descente	0.900	0,928	0.898
Galaxy-X - 2 : Recherche exhaustive	0.916	0,940	0.912
Galaxy-X -2 : L'algorithme de descente	0.730	0,785	0.724
Galaxy-X -2 : recherche paramètre par paramètre	0.916	0,94	0.912
Galaxy-X -2 : Optimisation Bayésienne	0.790	0,837	0.785

En particulier, nous avons constaté que l'algorithme de descente et l'optimisation bayésienne, bien que très efficaces pour des problèmes réputés difficiles, étaient sujettes à la "malédiction de la dimension", c'est-à-dire qu'elles peuvent devenir rapidement inefficaces, si tôt que la dimension devient trop importante.

Aussi, il reste un défi de taille : comment savoir si le modèle est généralisable et prédira bien dans l'avenir sur des données qui sont jamais vus auparavant ? Dans notre cas, l'un des plus grands problèmes dans l'apprentissage supervisé est le surapprentissage (Berrar, 2019), il est facile de construire un modèle parfaitement adapté à notre ensemble de données mais incapable de bien généraliser sur des

Tableau 5.2: La moyenne des résultats pour chaque algorithme avec la validation croisée comme méthode d'évaluation

	Précision	F-Mesure	Rappel
Galaxy-X	0.821	0,790	0.818
Galaxy-X - 1 : Recherche exhaustive	0.864	0,837	0.860
Galaxy-X -1 : L'algorithme de descente	0.798	0,790	0.793
Galaxy-X - 2 : Recherche exhaustive	0.872	0,861	0.870
Galaxy-X -2 : L'algorithme de descente	0.747	0,737	0.745
Galaxy-X -2 : recherche paramètre par paramètre	0.872	0,861	0.870
Galaxy-X -2 : Optimisation Bayésienne	0.827	0,793	0.825

données non vues auparavant.

Généraliser signifie aller de quelque chose de spécifique à quelque chose de large. Cette approche de généralisation nécessite que les données que nous utilisons pour former le modèle M constituent un bon échantillon fiable des observations que nous souhaitons que l'algorithme apprenne. Plus la qualité est élevée, plus elle est représentative et plus il sera facile pour le modèle d'apprendre.

Dans ce travail, nous avons utilisé la validation croisée pour échantiller les données. Cette méthode vise à éviter que le choix aléatoire des données d'entraînement par rapport aux données de test affecte nos attentes en matière de prédiction et à éviter le surapprentissage (Hastie *et al.*, 2009, Duda *et al.*, 2000). En d'autres

termes, si nous avons eu de la chance et avons choisi une bonne combinaison de formation et de validation parmi l'ensemble de données et que cette combinaison a donné d'excellentes prédictions sur les données de validation, les prédictions futures ne seraient probablement pas aussi réussis. Pour résoudre ce problème, plusieurs répétitions sont exécutées et moyennées (Simon, 2007).

Molinaro *et al.* (2005) ont comparé diverses méthodes d'échantillonnage de données de grande dimension, utilisées en bioinformatique. Leurs résultats suggèrent que la validation croisée par 10, et le bootstrap ont le plus petit biais. Cependant, il n'est pas clair quelle valeur de k doit être choisi pour la validation croisée des k-pli. Un choix judicieux est probablement $k = 10$, étant donné que l'estimation de l'erreur de prédiction est presque non biaisée dans une validation croisée multipliée par 10 (Simon, 2007). Isaksson *et al.* (2008) Soulignent toutefois que les mesures de performance validées par des méthodes croisées ne sont pas fiables pour les ensembles de données à petit échantillon.

Gràce à cette méthode d'échantillonage, les résultats que nous présentons dans ce mémoire sont considérés bien généralisable et comme une bonne estimation de la performance permettant de prédire les résultats futurs. Cependant, comme nous l'avons appliqué avec uniquement cinq plis et pour avoir un meilleur indice de généralisation, il sera préférable de tester les données de nouveau avec une validation croisée à dix plis.

CHAPITRE VI

CONCLUSION

L'apprentissage automatique est peut-être le plus utile pour l'interprétation de grands ensembles de données génomiques et a été utilisé pour annoter une grande variété de séquences génomiques.

Les algorithmes d'apprentissage automatique tentent de construire des modèles qui capturent un élément d'intérêt basé sur des données ce qui permet aux ordinateurs de faire et d'améliorer les prévisions ou les comportements. Malgré leur popularité et leur utilité, la plupart des techniques d'apprentissage automatique requièrent des connaissances approfondies pour guider le choix du modèle et des paramètres les mieux adaptés à un problème particulier. Dans de nombreux cas, ces connaissances spécialisées ne sont pas facilement disponibles. Quand c'est le cas, la complexité du problème et la subjectivité de l'expert peuvent souvent conduire à des choix sous-optimaux.

Le choix des hyperparamètres peut affecter de manière significative les performances du modèle résultant, mais la détermination de ces valeurs peut être complexes. Par conséquent, une stratégie de recherche disciplinée et théoriquement solide est essentielle.

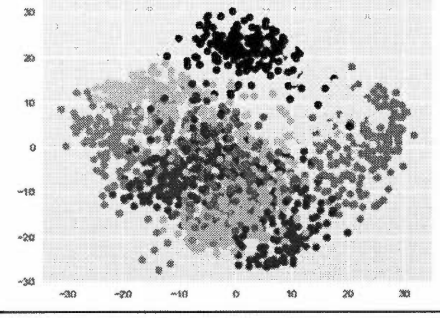
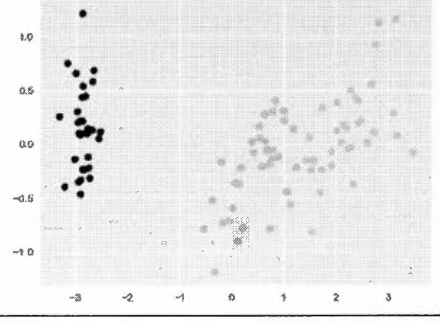
Nous avons présenté dans ce travail une méthode pour guider efficacement la

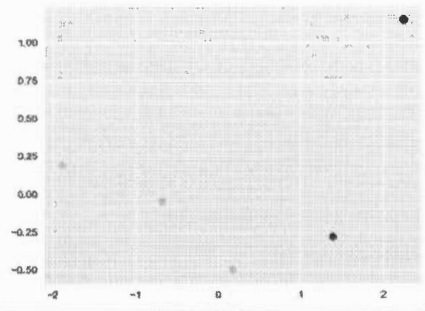
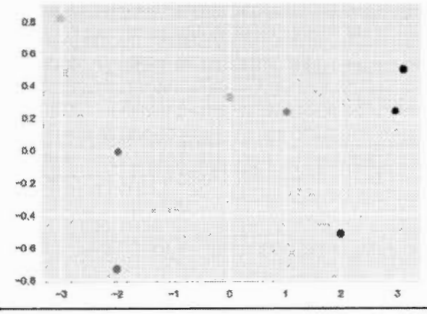
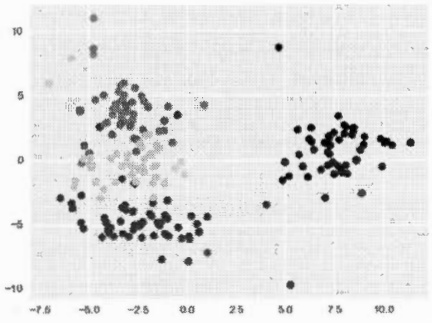
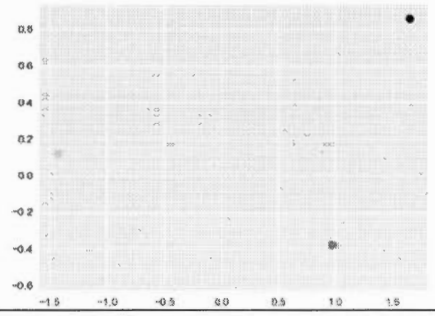
recherche de solutions optimales et le choix des hyperparamètres. Les résultats obtenus dans ce travail montrent que l'utilisation de la technique d'apprentissage automatique et de la configuration suggérées par notre approche automatisée donnent des prévisions de qualité bien supérieure aux résultats obtenus par une paramétrisation par défaut. La recherche exhaustive arrive toujours à trouver la solution optimale mais souffre de la malédiction de la dimensialité. L'optimisation bayésienne et l'algorithme de descente fonctionnent très bien sur quelques jeux de données et très mal sur d'autres. Ils dépendent du point de départ et ne trouvent pas toujours les scores globaux maximaux. Surtout l'optimisation bayésienne nécessite un nombre élevé d'itérations pour donner un bon rendement. Ils améliorent les résultats mais restent moins efficaces que la recherche exhaustive. Notre méthode 'la recherche paramètre par paramètre' a donné des résultats meilleurs que l'optimisation bayésienne et l'algorithme de descente mais doit être testé sur plus de données et avec d'autres types d'algorithmes autres que Galaxy-X.

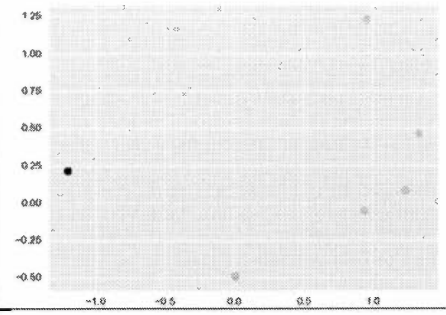
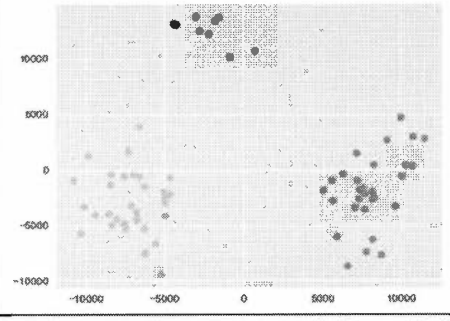
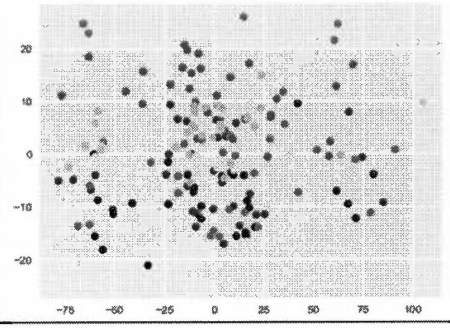
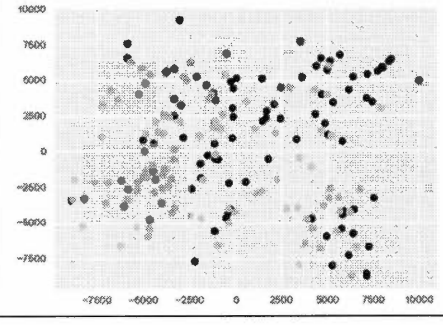
APPENDICE A

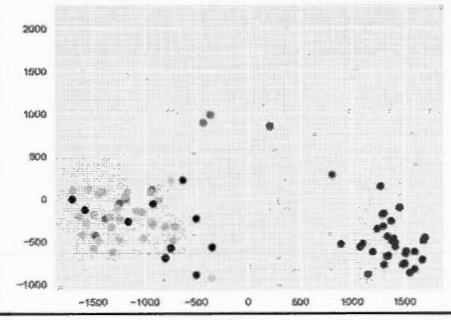
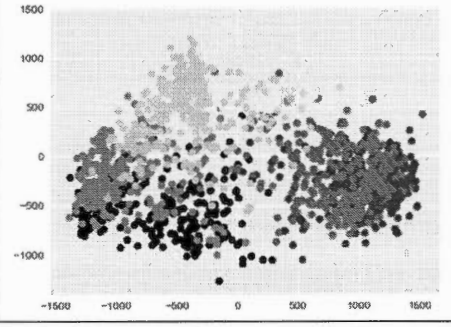
JEUX DE DONNÉES

Tableau A.1: Détails des jeux de données

Données	PCA	Nombre d'instances	Nombre de classes
digits		1797	10
iris		100	3

Données	PCA	Nombre d'instances	Nombre de classes
EBOSPECG		88	5
GEMGENCG		299	7
HCVGENCG		284	6
HIVGRPCG		76	4

Données	PCA	Nombre d'instances	Nombre de classes
ROTSPECG		146	3
PMVHIVGC1		76	4
PMVHIVGC2		189	6
PMVHIVGC5		200	2

Données	PCA	Nombre d'instances	Nombre de classes
PMVHIVPL1		94	6
PMVHIVPL2		1800	6

APPENDICE B

CODE SOURCE DE L'APPLICATION

B.1 Version originale de Galaxy-X

```
import time

def old_galaxy(X_train, X_test, y_train,
               y_test, old_softening = -0.4):
    start_time = time.clock() #Lancer le compteur
    galaxyX = old__galaxy.GalaxyX(local_estimator=classifier,
                                  softening=old_softening,
                                  disatnce_measure=distance)
    galaxyX.fit(X_train, y_train)
    predicted_labels = galaxyX.predict(X_test)
    recall, precision,
    TPR, TNR, PPV, NPV, FPR, FNR, FDR, ACC,
    score, f_scores, f_scores_classes,
    rejection_recall, rejection_precision = evaluation_metrics.
    evaluate_and_get_results(y_test, predicted_labels)
    runtime = time.clock() - start_time
    runtime_ = format(runtime, '.2f')
    return old_softening, runtime_, recall, precision,
        TPR, TNR, PPV, NPV, FPR, FNR, FDR, ACC,
        score, f_scores, f_scores_classes,
```

```
20          rejection_recall, rejection_precision
```

Listing B.1: La fonction qui exécute Galaxy-X

```
1  import numpy as np
2  import scipy.spatial.distance as d
3  from sklearn.base import BaseEstimator, ClassifierMixin
4  import pickle
5
6  class GalaxyX(BaseEstimator, ClassifierMixin):
7      def __init__(self, local_estimator = 'KNN', disatnce_measure = '
       euclidean', softening=0.0):
8          self.softening = softening
9          self.disatnce_measure = disatnce_measure
10         self.local_estimator = local_estimator
11         self.local_closed_set_classifier()
12         return
13     def local_closed_set_classifier(self):
14         if self.local_estimator == "KNN":
15             from sklearn import neighbors
16             self.local_estimator = neighbors.KNeighborsClassifier(
       n_neighbors=3, algorithm='brute', metric=self.disatnce_measure)
17         elif self.local_estimator == "SVM_Linear":
18             from sklearn.svm import SVC
19             self.local_estimator = SVC(kernel="linear")
20         elif self.local_estimator == "SVM_RBF":
21             from sklearn.svm import SVC
22             self.local_estimator = SVC()
23         elif self.local_estimator == "DT":
24             from sklearn.tree import DecisionTreeClassifier
25             self.local_estimator = DecisionTreeClassifier(
       random_state=0)
26         elif self.local_estimator == "Random_Forest":
27             from sklearn.ensemble import RandomForestClassifier
```

```
28              self.local_estimator = RandomForestClassifier()
29          elif self.local_estimator == "AdaBoost":
30              from sklearn.ensemble import AdaBoostClassifier
31              self.local_estimator = AdaBoostClassifier()
32          elif self.local_estimator == "NB":
33              from sklearn.naive_bayes import GaussianNB
34              self.local_estimator = GaussianNB()
35          elif self.local_estimator == "LDA":
36              from sklearn.lda import LDA
37              self.local_estimator = LDA()
38          elif self.local_estimator == "QDA":
39              from sklearn.qda import QDA
40              self.local_estimator = QDA()
41          else:
42              print("Unknown Classifier, set to default: KNN")
43              from sklearn import neighbors
44              self.local_estimator = neighbors.KNeighborsClassifier(
    n_neighbors=3, algorithm='brute', metric=self.disatnce_measure)
45          return self
46      def fit(self, X, y):
47          self.classes_, self.indices_ = np.unique(y, return_inverse=
    True)
48          self.saved_local_models_names = []
49          self.saved_local_models = {}
50          print('fit')
51          self.create_groups(X, y)
52          self.maxd_meanv()
53          return self
54      def create_groups(self, X, y):
55          self.groups = {}
56          print('create_groups')
57          for i in self.classes_:
58              self.groups[i]=[]
```

```
59          for i,v in enumerate(y):
60              self.groups[v].append(X[i])
61          return
62      def maxd_meanv(self):
63          self.center_radius={}
64          print('maxd_meanv')
65          for i in self.classes_ :
66              meanV = np.mean(self.groups[i], axis=0)
67              maxD=np.max(d.cdist([meanV],self.groups[i], self.
    disatnce_measure))
68              print('add softining value')
69              maxD+=(maxD * self.softening)
70              self.center_radius[i]=(maxD,meanV)
71          return
72      def Separability(self):
73          self.separability = 0
74          sum_sep = 0
75          l=list(set(self.center_radius))
76          nb_combinations = 0
77          if(len(l)<=1):
78              self.separability = 1
79          else:
80              for i in range(len(l)-1):
81                  (radiusi, meanVi) = self.center_radius[l[i]]
82                  for j in range(i+1,len(l)):
83                      nb_combinations+=1
84                      (radiusj, meanVj) = self.center_radius[l[j]]
85                      dist = float(d.cdist([meanVi],[meanVj], self.
    disatnce_measure)[0][0])
86                      if(dist >= (radiusi+radiusj)):
87                          sum_sep += 1
88                      elif(dist <= abs(radiusi-radiusj)):
89                          sum_sep += 0
```

```
90                  else:
91                      sum_sep += (dist/(radiusi + radiusj))
92          self.separability = (sum_sep/nb_combinations)
93      return
94  def predict(self, X):
95      predictions=[]
96      for test_vector in X:
97          predicted_label = self.filter_classes(test_vector)
98          if (len(predicted_label)<=0):
99              predicted_y = -1
100         elif (len(predicted_label)==1):
101             predicted_y = predicted_label[0]
102         else:
103             predicted_y = self.local_closed_classification(
    test_vector, predicted_label)
104         predictions.append(predicted_y)
105     return predictions
106 def filter_classes(self, X):
107     labels = []
108     for cls in list(self.groups.keys()):
109         if (float(d.cdist([X],[self.center_radius[cls][1]], self.
    disatnce_measure)[0][0]) <= float(self.center_radius[cls][0])):
110             labels.append(cls)
111     return labels
112 def local_closed_classification(self, x, labels):
113     name = "_".join(sorted(set(list(map(str,labels)))))
114     if (name in self.saved_local_models.keys()):
115         x = x.reshape(1, -1)
116         predicted_y = pickle.loads(self.saved_local_models[name
    ]).predict(x)[0]
117     else:
118         local_training_X, local_training_y = [], []
119         for i in labels:
```

```
120                 local_training_X+=list(self.groups[i])
121                 local_training_y += [i]*len(self.groups[i])
122             clf = self.local_estimator
123             clf.fit(local_training_X, local_training_y)
124             x = x.reshape(1, -1)
125             predicted_y = clf.predict(x)[0]
126             self.saved_local_models[name] = pickle.dumps(clf)
127         return predicted_y
128     def get_params(self, deep=True):
129         return {"softening": self.softening, "disatnce_measure": self
    .disatnce_measure, "local_estimator": self.local_estimator}
130     def set_params(self, **parameters):
131         for parameter, value in parameters.items():
132             self.setattr(parameter, value)
133         return self
```

Listing B.2: La classe Galaxy-X

B.2 Galaxy-X -1

```
1  import time
2
3  classifier = "SVM_Linear"
4  distance = 'euclidean'
5
6  def ext_grid_(X_train, X_test, y_train, y_test, combinations):
7      start_time = time.clock()
8      best_score = 0
9      softToKeep = 0
10     for soft in combinations:
11         soft, runtime_, recall, precision,
12         TPR, TNR, PPV, NPV, FPR, FNR, FDR, ACC,
13         score, f_scores, f_scores_classes,
14         rejection_recall, rejection_precision =
15             algorithmes_old.old_galaxy(
```

```
16                X_train, X_test, y_train, y_test, soft)
17            if (float(score) >= (best_score)):
18                best_score = float(score)
19                softToKeep = soft
20        soft, runtime_, recall, precision,
21        TPR, TNR, PPV, NPV, FPR, FNR, FDR, ACC,
22        score, f_scores, f_scores_classes,
23        rejection_recall, rejection_precision =
24            algorithmes_old.old_galaxy(
25            X_train, X_test, y_train, y_test, softToKeep)
26        runtime = time.clock() - start_time
27        runtime_ = format(runtime, '.2f')
28        return softToKeep, runtime_, recall, precision,
29            TPR, TNR, PPV, NPV, FPR, FNR, FDR, ACC,
30            score, f_scores, f_scores_classes,
31            rejection_recall, rejection_precision
```

Listing B.3: Recherche exhaustive

```
1  import time
2
3  classifier = "SVM_Linear"
4  distance = 'euclidean'
5
6  def hill_climbing(X_train, X_test, y_train, y_test):
7      start_time = time.clock()
8      soft, runtime_, recall, precision,
9      TPR, TNR, PPV, NPV, FPR, FNR, FDR, ACC,
10     score, f_scores, f_scores_classes,
11     rejection_recall, rejection_precision =
12         algorithmes_old.old_galaxy(
13         X_train, X_test, y_train, y_test)
14     steps = 30
15     best_softening = 0
```

```
    best_score = score
    x2 = - 0.45
    x3 = - 0.35
    for i in range(steps):
        soft, runtime_, recall, precision,
        TPR, TNR, PPV, NPV, FPR, FNR, FDR, ACC,
        scores2, f_scores, f_scores_classes,
        rejection_recall, rejection_precision =
            algorithmes_old.old_galaxy(
            X_train, X_test, y_train, y_test, x2)
        soft, runtime_, recall, precision,
        TPR, TNR, PPV, NPV, FPR, FNR, FDR, ACC,
        scores3, f_scores, f_scores_classes,
        rejection_recall, rejection_precision =
            algorithmes_old.old_galaxy(
            X_train, X_test, y_train, y_test, x3)
        if (x2 <= -1) or (x2 >= 1.0) or (x3 <= -1) or (x3 >= 1.0):
            break
        elif scores2 > best_score:
            x2 = x2 + 0.05
            best_score = scores2
            best_softening = x2
        elif scores3 > best_score:
            x3 = x3 + 0.05
            best_score = scores3
            best_softening = x3
        else:
            break
    soft, runtime_, recall, precision,
    TPR, TNR, PPV, NPV, FPR, FNR, FDR, ACC,
    score, f_scores, f_scores_classes,
    rejection_recall, rejection_precision =
        algorithmes_old.old_galaxy(
```

```
49          X_train, X_test, y_train, y_test, best_softening)
50      runtime = time.clock() - start_time
51      runtime_ = format(runtime, '.2f')
52      return soft, runtime_, recall, precision,
53          TPR, TNR, PPV, NPV, FPR, FNR, FDR, ACC,
54          score, f_scores, f_scores_classes,
55          rejection_recall, rejection_precision
```

Listing B.4: Algorithme de descente

B.3 Galaxy-X -2

```
1  import time
2
3  classifier = "SVM_Linear"
4  distance = 'euclidean'
5  softening = []
6
7  def ext_grid(X_train, X_test, y_train, y_test):
8      start_time = time.clock()
9      lenSoft = len(set(y_train))
10     for i in range(lenSoft):
11         softening.append(0)
12     softToKeep =
13         ExtGrid.ExtGrid(softening,
14             X_train, y_train, X_test, y_test,
15             classifier, distance).getBestGrid()
16     galaxyX =
17         galaxy.autoGalaxyX(local_estimator=classifier,
18         softening=softToKeep, disatnce_measure=distance)
19     galaxyX.fit(X_train, y_train)
20     predicted_labels = galaxyX.predict(X_test)
21     recall, precision,
22     TPR, TNR, PPV, NPV, FPR, FNR, FDR, ACC,
23     score, f_scores, f_scores_classes,
```

```
    rejection_recall, rejection_precision =
        evaluation_metrics.evaluate_and_get_results(y_test,
            predicted_labels)
    runtime = time.clock() - start_time
    runtime_ = format(runtime, '.2f')
    return softToKeep, runtime_, recall, precision,
        TPR, TNR, PPV, NPV, FPR, FNR, FDR, ACC,
        score, f_scores, f_scores_classes,
        rejection_recall, rejection_precision
```

Listing B.5: Recherche exhaustive

```
import time

classifier = "SVM_Linear"
distance = 'euclidean'
softening = []

def hill_climbing(X_train, X_test, y_train, y_test):
    start_time = time.clock()
    lenSoft = len(set(y_train))
    for i in range(lenSoft):
        softening.append(0)
    steps = 30
    best_softening = []
    best_score = 0
    x2 = - 0.15
    x3 = - 0.25
    softening2 = []
    softening3 = []
    for i in range(steps):
        for i in range(lenSoft):
            softening2.append(x2)
            softening3.append(x3)
```

```
23          galaxyX =
24              galaxy.autoGalaxyX(local_estimator=classifier,
25                  softening=softening2, disatnce_measure=distance)
26          galaxyX.fit(X_train, y_train)
27          predicted_labels = galaxyX.predict(X_test)
28          scores2 =
29              metrics.accuracy_score(y_test, predicted_labels)
30          galaxyX =
31              galaxy.autoGalaxyX(local_estimator=classifier,
32                  softening=softening3,
33                  disatnce_measure=distance)
34          galaxyX.fit(X_train, y_train)
35          predicted_labels = galaxyX.predict(X_test)
36          scores3 = metrics.accuracy_score(y_test,predicted_labels)
37          if (x2 <= -1) or (x2 >= 1.0) or (x3 <= -1) or (x3 >= 1.0):
38              break
39          elif scores2 >= best_score:
40              x2 = x2 + 0.05
41              best_score = scores2
42              best_softening = softening2
43          elif scores3 >= best_score:
44              x3 = x3 + 0.05
45              best_score = scores3
46              best_softening = softening3
47          else:
48              break
49      galaxyX =
50          galaxy.autoGalaxyX(local_estimator=classifier,
51              softening=best_softening, disatnce_measure=distance)
52      galaxyX.fit(X_train, y_train)
53      predicted_labels = galaxyX.predict(X_test)
54      recall, precision,
55      TPR, TNR, PPV, NPV, FPR, FNR, FDR, ACC,
```

```
56      score, f_scores, f_scores_classes,
57      rejection_recall, rejection_precision =
58          evaluation_metrics.evaluate_and_get_results(y_test,
59              predicted_labels)
60      runtime = time.clock() - start_time
61      runtime_ = format(runtime, '.2f')
62      return best_softening, runtime_, recall, precision,
63          TPR, TNR, PPV, NPV, FPR, FNR, FDR, ACC,
64          score, f_scores, f_scores_classes,
65          rejection_recall, rejection_precision
```

Listing B.6: Algorithme de descente

```
1  import time
2
3  classifier = "SVM_Linear"
4  distance = 'euclidean'
5  softening = []
6
7  def opt_grid(X_train, X_test, y_train, y_test):
8      start_time = time.clock()
9      lenSoft = len(set(y_train))
10     for i in range(lenSoft):
11         softening.append(0)
12     tableau =
13         OptGrid.OptGrid(softening, X_train, y_train,
14             X_test, y_test,
15             classifier, distance).getBestGrid()
16     galaxyX =
17         galaxy.autoGalaxyX(local_estimator=classifier,
18             softening=tableau, disatnce_measure=distance)
19     galaxyX.fit(X_train, y_train)
20     predicted_labels = galaxyX.predict(X_test)
21     recall, precision,
```

```
22      TPR, TNR, PPV, NPV, FPR, FNR, FDR, ACC,
23      score, f_scores, f_scores_classes,
24      rejection_recall, rejection_precision =
25          evaluation_metrics.evaluate_and_get_results(
26          y_test, predicted_labels)
27      runtime = time.clock() - start_time
28      runtime_ = format(runtime, '.2f')
29      return tableau, runtime_, recall, precision,
30          TPR, TNR, PPV, NPV, FPR, FNR, FDR, ACC,
31          score, f_scores, f_scores_classes,
32          rejection_recall, rejection_precision
```

Listing B.7: La recherche paramètre par paramètre

```
1  from .bayes_opt import BayesianOptimization
2  import time
3
4  classifier = "SVM_Linear"
5  distance = 'euclidean'
6  softening = []
7
8  def bayes(X_train, X_test, y_train, y_test):
9      softening = []
10     BOparams = []
11     interval = (-0.9, -0.1)
12     d = {}
13     start_time = time.clock()
14     lenSoft = len(set(y_train))
15     for i in range(lenSoft):
16         softening.append(0)
17         BOparams.append(chr(97+i))
18         d[chr(97+i)] = interval
19     gp_params = {"alpha": 1e-5}
20     def algoToMax():
```

```
        galaxyX =
            galaxy.autoGalaxyX(local_estimator=classifier,
                softening=x, disatnce_measure=distance)
        galaxyX.fit(X_train, y_train)
        predicted_labels = galaxyX.predict(X_test)
        scores =
            metrics.accuracy_score(y_test, predicted_labels)
        return scores
    galaxyBO = BayesianOptimization(algoToMax, d)
    galaxyBO.maximize(n_iter=10, **gp_params)
    y = galaxyBO.res['max']['max_params']
    final_list = []
    for key, value in y.items():
        final_list.append(value)
    galaxyX =
        galaxy.autoGalaxyX(local_estimator=classifier,
            softening=final_list, disatnce_measure=distance)
    galaxyX.fit(X_train, y_train)
    predicted_labels = galaxyX.predict(X_test)
    recall, precision,
    TPR, TNR, PPV, NPV, FPR, FNR, FDR, ACC,
    score, f_scores, f_scores_classes,
    rejection_recall, rejection_precision =
        evaluation_metrics.evaluate_and_get_results(
            y_test, predicted_labels)
    runtime = time.clock() - start_time
    runtime_ = format(runtime, '.2f')
    return final_list, runtime_, recall, precision,
        TPR, TNR, PPV, NPV, FPR, FNR, FDR, ACC,
        score, f_scores, f_scores_classes,
        rejection_recall, rejection_precision
```

Listing B.8: L'optimisation bayésienne

RÉFÉRENCES

Abergel, C., Legendre, M. et Claverie, J.-M. (2015). The rapidly expanding universe of giant viruses : Mimivirus, Pandoravirus, Pithovirus and Mollivirus. *FEMS microbiology reviews*, *39*(6), 779–796. http://dx.doi.org/10.1093/femsre/fuv037

Aggarwal, C. C. (2013). *Outlier Analysis*. New York : Springer-Verlag. Récupéré le 2018-11-29 de //www.springer.com/us/book/9781461463955

Altschul, S. F., Gish, W., Miller, W., Myers, E. W. et Lipman, D. J. (1990). Basic local alignment search tool. *Journal of Molecular Biology*, *215*(3), 403–410. http://dx.doi.org/10.1016/S0022-2836(05)80360-2

Altschul, S. F., Madden, T. L., Schäffer, A. A., Zhang, J., Zhang, Z., Miller, W. et Lipman, D. J. (1997). Gapped BLAST and PSI-BLAST : a new generation of protein database search programs. *Nucleic Acids Research*, *25*(17), 3389–3402.

Amiri, S. et Dinov, I. D. (2016). Comparison of genomic data via statistical distribution. *Journal of Theoretical Biology*, *407*, 318–327. http://dx.doi.org/10.1016/j.jtbi.2016.07.032

Attwood, T. K. (2000). Genomics. The Babel of bioinformatics. *Science (New York, N.Y.)*, *290*(5491), 471–473.

Bao, Y., Chetvernin, V. et Tatusova, T. (2014). Improvements to pairwise sequence comparison (PASC) : a genome-based web tool for virus classification. *Archives of Virology*, *159*(12), 3293–3304. http://dx.doi.org/10.1007/s00705-014-2197-x

Bergstra, J. et Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, *13*(Feb), 281–305. Récupéré le 2019-01-05 de http://www.jmlr.org/papers/v13/bergstra12a.html

Bergstra, J. S., Bardenet, R., Bengio, Y. et Kégl, B. (2011). Algorithms for Hyper-Parameter Optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, et K. Q. Weinberger (dir.), *Advances in Neural Information Processing Systems 24* 2546–2554. Curran Associates, Inc.

Berrar, D. (2019). Cross-Validation. In S. Ranganathan, M. Gribskov, K. Nakai, et C. Schönbach (dir.), *Encyclopedia of Bioinformatics and Computational Biology* 542–545. Oxford : Academic Press

Bichaud, L., de Lamballerie, X., Alkan, C., Izri, A., Gould, E. A. et Charrel, R. N. (2014). Arthropods as a source of new RNA viruses. *Microbial Pathogenesis*, *77*, 136–141. http://dx.doi.org/10.1016/j.micpath.2014.09.002

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg : Springer-Verlag.

Brochu, E., Cora, V. M. et de Freitas, N. (2010). A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. *arXiv :1012.2599 [cs]*. arXiv : 1012.2599. Récupéré le 2019-01-05 de http://arxiv.org/abs/1012.2599

Bronze, M. S., Huycke, M. M., Machado, L. J., Voskuhl, G. W. et Greenfield, R. A. (2002). Viral agents as biological weapons and agents of bioterrorism. *The American Journal of the Medical Sciences*, *323*(6), 316–325.

Cano, A., Gómez-Olmedo, M. et Moral, S. (2005). Application of a hill-climbing algorithm to exact and approximate inference in credal networks. 88–97.

Claesen, M. et De Moor, B. (2015). Hyperparameter Search in Machine Learning. *arXiv :1502.02127 [cs, stat]*. arXiv : 1502.02127. Récupéré le 2019-01-05 de http://arxiv.org/abs/1502.02127

Coates, A., Ng, A. Y. et Lee, H. (2011). An Analysis of Single-Layer Networks in Unsupervised Feature Learning. Dans *AISTATS*.

Cooper, P. W. (1962). The hypersphere in pattern recognition. *Information and Control*, *5*(4), 324–346. http://dx.doi.org/10.1016/S0019-9958(62)90641-1. Récupéré le 2019-01-05 de http://www.sciencedirect.com/science/article/pii/S0019995862906411

Cover, T. et Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, *13*(1), 21–27. http://dx.doi.org/10.1109/TIT.1967.1053964. Récupéré le 2019-01-01 de http://ieeexplore.ieee.org/document/1053964/

Dayhoff, M. O. et Schwartz, R. M. (1978). Chapter 22 : A model of evolutionary change in proteins. Dans *in Atlas of Protein Sequence and Structure*.

Delcher, A. L., Kasif, S., Fleischmann, R. D., Peterson, J., White, O. et Salzberg, S. L. (1999). Alignment of whole genomes. *Nucleic Acids Research*, *27*(11), 2369–2376.

Dhifli, W. et Diallo, A. (2016). Toward an Efficient Multi-class Classification in an Open Universe.

Duda, R. O., Hart, P. E. et Stork, D. G. (2000). *Pattern Classification (2Nd Edition)*. New York, NY, USA : Wiley-Interscience.

Durbin, R., Eddy, S. R., Krogh, A. et Mitchison, G. (1998). Biological Sequence Analysis by Richard Durbin. http://dx.doi.org/10.1017/CBO9780511790492. Récupéré le 2019-01-05 de /core/books/biological-sequence-analysis/921BB7B78B745198829EF96BC7E0F29D

Edgar, R. C. (2004). MUSCLE : multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, *32*(5), 1792–1797. http://dx.doi.org/10.1093/nar/gkh340

Escalante, A. E., Jardón Barbolla, L., Ramírez-Barahona, S. et Eguiarte, L. E. (2014). The study of biodiversity in the era of massive sequencing. *Revista Mexicana de Biodiversidad*, *85*(4), 1249–1264. http://dx.doi.org/10.7550/rmb.43498. Récupéré le 2018-11-29 de http://www.sciencedirect.com/science/article/pii/S1870345314730076

Gelderblom, H. R. (1996). Structure and Classification of Viruses. In S. Baron (dir.), *Medical Microbiology*. Galveston (TX) : University of Texas Medical Branch at Galveston, (4th éd.)

han, J., Pei, J. et Kamber, M. (2011). Data Mining : Concepts and Techniques - 3rd Edition. Récupéré le 2019-01-07 de https://www.elsevier.com/books/data-mining-concepts-and-techniques/han/978-0-12-381479-1

Hastie, T., Tibshirani, R. et Friedman, J. (2009). *The Elements of Statistical Learning : Data Mining, Inference, and Prediction, Second Edition* (2 éd.). Springer Series in Statistics. New York : Springer-Verlag. Récupéré le 2019-06-05 de https://www.springer.com/gp/book/9780387848570

Hawkins, D. (1980). *Identification of Outliers*. Monographs on Statistics and Applied Probability. Springer Netherlands. Récupéré le 2019-01-05 de //www.springer.com/la/book/9789401539968

Henikoff, S. et Henikoff, J. G. (1992). Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences of the United States of America*, *89*(22), 10915–10919.

Henson, J., Tischler, G. et Ning, Z. (2012). Next-generation sequencing and large genome assemblies. *Pharmacogenomics*, *13*(8), 901–915. http://dx.doi.org/10.2217/pgs.12.72

Hernandez, T. et Yang, J. (2016). Descriptive Statistics of the Genome : Phylogenetic Classification of Viruses. *Journal of Computational Biology : A Journal of Computational Molecular Cell Biology*, *23*(10), 810–820. http://dx.doi.org/10.1089/cmb.2013.0132

Ho, T. K. (1995). Random Decision Forests. Dans *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*, ICDAR '95, 278–., Washington, DC, USA. IEEE Computer Society. Récupéré le 2019-01-01 de http://dl.acm.org/citation.cfm?id=844379.844681

Hutter, F., Hoos, H. et Leyton-Brown, K. (2014). An Efficient Approach for Assessing Hyperparameter Importance. Dans *International Conference on Machine Learning*, 754–762. Récupéré le 2019-01-05 de http://proceedings.mlr.press/v32/hutter14.html

Hutter, F., Hoos, H. H. et Leyton-Brown, K. (2011). Sequential Model-based Optimization for General Algorithm Configuration. Dans *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, LION'05, 507–523., Berlin, Heidelberg. Springer-Verlag. http://dx.doi.org/10.1007/978-3-642-25566-3_40. Récupéré le 2019-01-05 de http://dx.doi.org/10.1007/978-3-642-25566-3_40

Isaksson, A., Wallman, M., Göransson, H. et Gustafsson, M. G. (2008). Cross-validation and Bootstrapping Are Unreliable in Small Sample Classification. *Pattern Recogn. Lett.*, *29*(14), 1960–1965. http://dx.doi.org/10.1016/j.patrec.2008.06.018. Récupéré le 2019-06-05 de http://dx.doi.org/10.1016/j.patrec.2008.06.018

Jones, D. R., Schonlau, M. et Welch, W. J. (1998). Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, *13*(4), 455–492. http://dx.doi.org/10.1023/A:1008306431147. Récupéré le 2019-01-05 de https://doi.org/10.1023/A:1008306431147

Just, W. (2001). Computational complexity of multiple sequence alignment with SP-score. *Journal of Computational Biology : A Journal of Computational Molecular Cell Biology*, *8*(6), 615–623. http://dx.doi.org/10.1089/106652701753307511

Katoh, K., Misawa, K., Kuma, K.-i. et Miyata, T. (2002). MAFFT : a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Research*, *30*(14), 3059–3066.

Kaur, Y. et Sohi, N. (2017). Comparison of Different Sequence Alignment Methods- A Survey. *International Journal of Advanced Research in Computer Science*, *8*(5), 2308–2311. http://dx.doi.org/10.26483/ijarcs.v8i5.3916. Récupéré le 2019-01-05 de http://www.ijarcs.info/index.php/Ijarcs/article/view/3916

Larkin, M. A., Blackshields, G., Brown, N. P., Chenna, R., McGettigan, P. A., McWilliam, H., Valentin, F., Wallace, I. M., Wilm, A., Lopez, R., Thompson, J. D., Gibson, T. J. et Higgins, D. G. (2007). Clustal W and Clustal X version 2.0. *Bioinformatics (Oxford, England)*, *23*(21), 2947–2948. http://dx.doi.org/10.1093/bioinformatics/btm404

Lebatteux, D., Remita, A. M. et Diallo, A. B. (2019). Toward an Alignment-Free Method for Feature Extraction and Accurate Classification of Viral Sequences. *Journal of Computational Biology*, *26*(6), 519–535. http://dx.doi.org/10.1089/cmb.2018.0239. Récupéré le 2019-06-17 de https://www.liebertpub.com/doi/full/10.1089/cmb.2018.0239

Leslie, C. S., Eskin, E., Cohen, A., Weston, J. et Noble, W. S. (2004). Mismatch string kernels for discriminative protein classification. *Bioinformatics (Oxford, England)*, *20*(4), 467–476. http://dx.doi.org/10.1093/bioinformatics/btg431

Liu, L., Li, Y., Li, S., Hu, N., He, Y., Pong, R., Lin, D., Lu, L. et Law, M. (2012). Comparison of Next-Generation Sequencing Systems. *Journal of Biomedicine and Biotechnology*, *2012*. http://dx.doi.org/10.1155/2012/251364. Récupéré le 2019-01-05 de https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3398667/

Lynch, M. (2002). Intron evolution as a population-genetic process. *Proceedings of the National Academy of Sciences of the United States of America*, *99*(9), 6118–6123. http://dx.doi.org/10.1073/pnas.092595699. Récupéré le 2018-11-29 de https://www.ncbi.nlm.nih.gov/pmc/articles/PMC122912/

Mahmoudabadi, G. et Phillips, R. (2018). A comprehensive and quantitative exploration of thousands of viral genomes. *eLife*, *7*, e31955. http://dx.doi.org/10.7554/eLife.31955. Récupéré le 2019-01-02 de https://doi.org/10.7554/eLife.31955

Mardis, E. R. (2008). Next-generation DNA sequencing methods. *Annual Review of Genomics and Human Genetics*, *9*, 387–402. http://dx.doi.org/10.1146/annurev.genom.9.081307.164359

Molinaro, A. M., Simon, R. et Pfeiffer, R. M. (2005). Prediction error estimation : a comparison of resampling methods. *Bioinformatics (Oxford, England)*, *21*(15), 3301–3307. http://dx.doi.org/10.1093/bioinformatics/bti499

Muhire, B., Martin, D. P., Brown, J. K., Navas-Castillo, J., Moriones, E., Zerbini, F. M., Rivera-Bustamante, R., Malathi, V. G., Briddon, R. W. et Varsani, A. (2013). A genome-wide pairwise-identity-based proposal for the classification of viruses in the genus Mastrevirus (family Geminiviridae). *Archives of Virology*, *158*(6), 1411–1424. http://dx.doi.org/10.1007/s00705-012-1601-7

Muhire, B. M., Varsani, A. et Martin, D. P. (2014). SDT : a virus classification tool based on pairwise sequence alignment and identity calculation. *PloS One*, *9*(9), e108277. http://dx.doi.org/10.1371/journal.pone.0108277

Needleman, S. B. et Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, *48*(3), 443–453.

Pinto, N., Doukhan, D., DiCarlo, J. J. et Cox, D. D. (2009). A High-Throughput Screening Approach to Discovering Good Forms of Biologically Inspired Visual Representation. *PLOS Computational Biology*, *5*(11), e1000579. http://dx.doi.org/10.1371/journal.pcbi.1000579. Récupéré le 2018-12-23 de https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1000579

Rasmussen, C. E. et Williams, C. K. I. (2005). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Remita, M. A., Halioui, A., Malick Diouara, A. A., Daigle, B., Kiani, G. et Diallo, A. B. (2017). A machine learning approach for viral genome classification. *BMC bioinformatics*, *18*(1), 208. http://dx.doi.org/10.1186/s12859-017-1602-3

Rocha, A. et Klein Goldenstein, S. (2014). Multiclass from Binary : Expanding One-Versus-All, One-Versus-One and ECOC-Based Approaches. *IEEE transactions on neural networks and learning systems*, *25*, 289–302. http://dx.doi.org/10.1109/TNNLS.2013.2274735

Safavian, S. R. et Landgrebe, D. (1991). A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man, and Cybernetics*, *21*(3), 660–674. http://dx.doi.org/10.1109/21.97458

Scheirer, W. J., de Rezende Rocha, A., Sapkota, A. et Boult, T. E. (2013). Toward open set recognition. *IEEE transactions on pattern analysis and machine intelligence*, *35*(7), 1757–1772. http://dx.doi.org/10.1109/TPAMI.2012.256

Scholkopf, B. et Smola, A. J. (2001). *Learning with Kernels : Support Vector Machines, Regularization, Optimization, and Beyond.* Cambridge, MA, USA : MIT Press.

She, R., Chen, F., Wang, K., Ester, M., Gardy, J. L. et Brinkman, F. S. L. (2003). Frequent-subsequence-based Prediction of Outer Membrane Proteins. Dans *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, 436–445., New York, NY, USA. ACM. http://dx.doi.org/10.1145/956750.956800. Récupéré le 2018-11-29 de http://doi.acm.org/10.1145/956750.956800

Simmonds, P., Adams, M. J., Benkő, M., Breitbart, M., Brister, J. R., Carstens, E. B., Davison, A. J., Delwart, E., Gorbalenya, A. E., Harrach, B., Hull, R., King, A. M. Q., Koonin, E. V., Krupovic, M., Kuhn, J. H., Lefkowitz, E. J., Nibert, M. L., Orton, R., Roossinck, M. J., Sabanadzovic, S., Sullivan, M. B., Suttle, C. A., Tesh, R. B., van der Vlugt, R. A., Varsani, A. et Zerbini, F. M. (2017). Consensus statement : Virus taxonomy in the age of metagenomics. *Nature Reviews. Microbiology*, *15*(3), 161–168. http://dx.doi.org/10.1038/nrmicro.2016.177

Simmonds, P. et Aiewsakun, P. (2018). Virus classification - where do you draw the line? *Archives of Virology*, *163*(8), 2037–2046. http://dx.doi.org/10.1007/s00705-018-3938-z

Simmonds, P., Tulloch, F., Evans, D. J. et Ryan, M. D. (2015). Attenuation of dengue (and other RNA viruses) with codon pair recoding can be explained by increased CpG/UpA dinucleotide frequencies. *Proceedings of the National Academy of Sciences of the United States of America*, *112*(28), E3633–3634. http://dx.doi.org/10.1073/pnas.1507339112

Simon, R. (2007). Resampling Strategies for Model Assessment and Selection. In W. Dubitzky, M. Granzow, et D. Berrar (dir.), *Fundamentals of Data Mining in Genomics and Proteomics* 173–186. Boston, MA : Springer US

Snoek, J., Larochelle, H. et Adams, R. P. (2012). Practical Bayesian Optimization of Machine Learning Algorithms. *arXiv :1206.2944 [cs, stat]*. arXiv : 1206.2944. Récupéré le 2019-01-05 de http://arxiv.org/abs/1206.2944

Tax, D. (2001). One-Class Classification ; Concept-Learning In The Absence Of Counter-Examples.

Thornton, C., Hutter, F., Hoos, H. H. et Leyton-Brown, K. (2012). Auto-WEKA : Combined Selection and Hyperparameter Optimization of Classification Algorithms. *arXiv :1208.3719 [cs]*. arXiv : 1208.3719. Récupéré le 2018-12-23 de http://arxiv.org/abs/1208.3719

Tomović, A., Janičić, P. et Kešelj, V. (2006). n-Gram-based classification and unsupervised hierarchical clustering of genome sequences. *Computer Methods and Programs in Biomedicine*, *81*(2), 137–153. http://dx.doi.org/10.1016/j.cmpb.2005.11.007. Récupéré le 2019-01-07 de http://www.sciencedirect.com/science/article/pii/S0169260705002361

Torralba, A. et Efros, A. A. (2011). Unbiased look at dataset bias. Dans *in CVPR*.

Vinga, S. et Almeida, J. (2003). Alignment-free sequence comparison-a review. *Bioinformatics (Oxford, England)*, *19*(4), 513–523.

Waterman, M. S. (1995). Introduction to computational biology - maps, sequences, and genomes : interdisciplinary statistics.

Waye, M. M. Y. et Sing, C. W. (2010). Anti-Viral Drugs for Human Adenoviruses. *Pharmaceuticals*, *3*(10), 3343–3354. http://dx.doi.org/10.3390/ph3103343. Récupéré le 2019-01-02 de https://www.mdpi.com/1424-8247/3/10/3343

Wolpert, D. et Macready, W. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67–82. http://dx.doi.org/10.1109/4235.585893. Récupéré le 2019-06-17 de http://ieeexplore.ieee.org/document/585893/

Xing, Z., Pei, J. et Keogh, E. (2010). A brief survey on sequence classification. *ACM SIGKDD Explorations Newsletter*, *12*(1), 40. http://dx.doi.org/10.1145/1882471.1882478. Récupéré le 2018-12-23 de http://portal.acm.org/citation.cfm?doid=1882471.1882478

Yu, C., Hernandez, T., Zheng, H., Yau, S.-C., Huang, H.-H., He, R. L., Yang, J. et Yau, S. S.-T. (2013). Real time classification of viruses in 12 dimensions. *PloS One*, *8*(5), e64328. http://dx.doi.org/10.1371/journal.pone.0064328

Yu, C., Liang, Q., Yin, C., He, R. L. et Yau, S. S.-T. (2010). A novel construction of genome space with biological geometry. *DNA research : an international journal for rapid publication of reports on genes and genomes*, *17*(3), 155–168. http://dx.doi.org/10.1093/dnares/dsq008

Zhang, Y., Yu, Z., Fu, X. et Liang, C. (2002). Noc3p, a bHLH protein, plays an integral role in the initiation of DNA replication in budding yeast. *Cell*, *109*(7), 849–860.