Parallèlisation et passage à l'échelle d'algorithmes de réduction de données pour le Datamining

Reine Marie Ndèla Marone

REPUBLIQUE DU SENEGAL

UNIVERSITE CHEIKH ANTA DIOP DE DAKAR

FACULTE DES SCIENCES ET TECHNIQUES

DEPARTEMENT DE MATHEMATIQUES ET INFORMATIQUE

THESE DE DOCTORAT UNIQUE

Ecole Doctorale Mathématiques et Informatique de l'UCAD

Spécialité : Informatique

N° d'ordre : **119**

Parallèlisation et passage à l'échelle d'algorithmes de réduction de données pour le Datamining

Présentée par :

Reine Marie Ndéla MARONE

Thèse présentée pour obtenir le grade universitaire de docteur

Soutenue le 07 juillet 2018 devant le jury composé de :

Hamidou	DATHE	Président (Professeur, Université Cheikh Anta Diop de Dakar)
Yahia	SLIMANI	Rapporteur (Professeur, Université Manouba, Tunisie)
Cheikh Talibouya	DIOP	Rapporteur (Professeur, Université Gaston Berger de Saint-Louis)
Karim	KONATE	Examinateur (Professeur, Université Cheikh Anta Diop de Dakar)
Samba	NDIAYE	Directeur de thèse (Maître de Conférences, Université Cheikh Anta Diop de Dakar)

Résumé

Une définition simple du BIG DATA : c'est un ensemble de données qui est si grand qu'il ne peut pas être traité en utilisant des méthodes conventionnelles. Le BIG DATA est caractérisé par les 5V (volume, variété, vélocité, véracité et valeur). Le volume se réfère à la quantité de données, la variété signifie que les données sont générées à partir de plusieurs sources dans plusieurs formats différents et la vitesse désigne la vitesse du traitement des données. La véracité quant à elle fait référence à la fiabilité des données. Et la valeur signifie une capacité à se concentrer sur les données qui ont une réelle valeur. Cependant ces grandes masses de données contiennent des informations redondantes ou non pertinentes qui peuvent altérer les performances des algorithmes de la Fouille de Données. La Fouille de Données ou Datamining est définie comme l'extraction de connaissances à partir des données. Ainsi, il est primordial d'éliminer ces données redondantes ou non pertinentes pour exploiter de façon efficiente le BIG DATA.

Malheureusement, les méthodes habituelles de réduction de données redondantes telles que la sélection d'attributs et la sélection d'instances permettent cette suppression mais ont des performances médiocres en termes d'exécution. Elles ne sont pas capables de faire face à la forte volumétrie des données BIG DATA. On dit alors qu'elles ne passent pas à l'échelle. En effet, elles ont été conçues pour une architecture informatique centralisée. Elles se révèlent incapables de profiter des nouvelles infrastructures composées de clusters de plusieurs milliers d'ordinateurs totalisant une capacité presque infinie de calcul et de stockage, pour traiter les données volumineuses du BIG DATA.

APACHE SPARK est un nouvel environnement de calcul distribué qui veut tirer parti de ces nouvelles infrastructures, pour traiter de façon plus efficiente les données du BIG DATA. Ce nouveau framework permet la parrallélisation des algorithmes de datamining en distribuant calculs et données sur les mémoires RAM des milliers d'ordinateurs des clusters.

Cette thèse étudie le couplage du datamining avec l'environnement APACHE SPARK. Elle s'intéresse plus particulièrement à la parallélisation des algorithmes de réduction de données dans le contexte du BIG DATA. Nous avons proposé des méthodes à large échelle de sélection d'attributs et d'instances dans l'environnement

APACHE SPARK. Nos expérimentations sur de gros volumes de données montrent l'efficacité de nos méthodes. De plus, celles-ci passent à l'échelle quand les données deviennent volumineuses.

Mots-clés : sélection d'attributs, méthodes enveloppantes, calcul parallèle, APACHE SPARK, SVM-RFE, SVM, méthodes filtres, données à large échelle, passage à l'échelle, BIG DATA.

Abstract

In simplification big data is a dataset, which is so large that it cannot be managed with the usual conventional methods. The 5V (volume, variety, velocity, veracity and value) are five proprieties or determinant dimensions of big data. Volume refers to the quantity of datas, variety means that the datas are generated from several sources in many different formats and velocity is the speed of the data treatment. As for veracity it refers to the reliability of the data. And value tells the capacity of concentrating on the datas with real value. However, those larges sets of data contain redundant and nonrelevant informations that can alter the performances of the datamining algorithms.

That's the reason why they must be cleaned away from the big datasets before any other operations. The data reduction methods such as feature or instance selection can help in their cleaning. But they cannot face with the large volumes of big data, because they have been conceived for centralized computing and not to be run on cluster machines. Therefore they are not adapted for large set. What is more, they, sometimes take enormous time to run.

Another fact is that the reduction of the cost of RAM memory has given birth to Apache Spark, a framework specialized in the analysis of big data, which can achieve in memory treatments using a cluster of machines. That environment can allow massive parallelization, which consist in distributing the treatments in great number of machines according to the size of your enterprise. Thus, the datamining technologies can use those apache spark advantages to adapt the datamining algorithms to the analysis of big data sets.

Our work here, studies the coupling of datamining with the environment of spark. It is interested in the reduction of the data, which is of great utility in big data, which is characterized by a low density of informations. For that reason, we have proposed in this memoire, large scale features and instances selection methods developed in the Spark environment. The experiments that we made on large-scale datasets show that our methods behave very efficiently when dealing with big data.

Key-words: feature selection, instance selection, wrapper method, parallel computing, apache spark, SVM-RFE, SVM, filter methods, big data, high dimensionality, large-datasets, scalability, large-scale.

.

Remerciements

Je commence par exprimer ma profonde gratitude au Professeur Samba NDIAYE, pour m'avoir fait l'honneur d'être mon directeur de thèse. Je le remercie aussi de m'avoir acceptée dans son équipe de recherche, me donnant ainsi l'occasion de travailler dans un domaine qui me passionne énormément.

Je tiens à remercier particulièrement le docteur Fodé CAMARA de l'UADB de Bambey, pour m'avoir soutenue tout au long de ma thèse. De plus son aide et ses conseils m'ont permis de rebondir dans les moments difficiles. Je le remercie vivement pour l'aide scientifique précieuse et tous les conseils qu'il m'a donnés pendant la durée de cette thèse.

Je remercie infiniment tous les membres du jury qui ont bien voulu consacrer du temps précieux à l'évaluation de mon travail. Je tiens particulièrement à remercier :

- Le Président du jury le Professeur Hamidou DATHE ;
- Les Professeurs Yahia SLIMANI et Cheikh Talibouya DIOP qui ont bien voulu être rapporteurs de cette thèse ;
- Le Professeur Karim KONATE qui a bien voulu participer au jury.

Je remercie infiniment le Professeur Mbaye SENE de l'UCAD qui a toujours été d'un grand soutien pour moi depuis le master. Il ne nous a jamais refusé son aide et a toujours été d'une grande accessibilité et générosité à notre égard.

A tous mes étudiants de toutes les écoles où je dispense des cours un grand merci pour votre soutien et vos encouragements.

Mes remerciements vont à l'endroit de tous mes amis d'enfance et tous ceux qui de près ou de loin m'ont soutenue.

J'aimerais remercier du fond du cœur mes parents pour leur soutien moral et matériel, ma famille qui a toujours porté un intérêt à ce que je faisais. Qu'elle trouve dans ce travail le témoignage de mon affection.

Dédicaces

À mes chers parents.

Publications

Les articles suivants, publiés à des proceeding de conférences IEEE et dans les journaux Springer, sont des résultats directs ou indirects de la recherche discutée dans cette thèse.

Conférences

Reine Marie Marone; Fodé Camara; Samba Ndiaye. A Parallelized mRMR Method for Feature Selection Based on Spark.8ème Edition de la Conférence Nationale sur la Recherche en Informatique et ses Applications (CNRIA). Du 18 au 21 avril 2018 à Ziguinchor. (Accepté pour publication dans Springer)

Reine Marie Marone; Fodé Camara; Samba Ndiaye. Two Spark Parallelized Methods for Feature Selection. AFRICATEK 2018 - 2nd EAI International Conference on Emerging Technologies for Developing Countries. 29 mai 2018. Cotonou, Benin. (Accepté pour publication dans Springer)

Demba Kandé, Fodé Camara, **Reine Marie Marone**, Samba Ndiaye. Vector Space Model of Text Classification based on Inertia Contribution of Document. AFRICATEK 2018 - 2nd EAI International Conference on Emerging Technologies for Developing Countries. 29 mai 2018. Cotonou, Benin. (Accepté pour publication dans Springer)

Demba Kandé, Fodé Camara, **Reine Marie Marone**, Samba Ndiaye. A Novel Term Weighting Scheme Model. 2018 3rd International Conference on Knowledge Engineering and Applications(ICKEA 2018). Moscow, Russia on June 25-27, 2018. (publication IEEE)

Reine Marie Marone; Fodé Camara; Samba Ndiaye. A large-scale filter method for feature selection based on spark. 2017 IEEE 4th International Conference on Soft Computing & Machine Intelligence (ISCMI).

Reine Marie Marone; Fodé Camara; Samba Ndiaye. LSIS: Large scale instance selection algorithm for big data. 2017 3rd IEEE International Conference on Computer and Communications (ICCC). (publication IEEE)

Table des matières

Table des figures

Introduction

1. Contexte et problématique

Les nouvelles technologies de l'information telles que le Web 2.0 ont contribué à la croissance exponentielle des données. Celles-ci affluent de plus en plus vite, presque en temps réel et sous un format très varié. Internet par exemple continue de générer des quintillions d'octets de données. En 2012 par exemple, deux et demi exaoctets de données ont été créés quotidiennement. Au cours des deux dernières années, la quantité de données produites a dépassé celle de toute l'histoire de l'humanité [1, 2, 3]. Le BIG DATA, est le terme qui a été inventé pour décrire cette croissance exponentielle des données. Le BIG DATA a souvent été défini par les 3V [4, 5, 6] qui sont le volume, la vitesse et la variété, auxquelles sont venus s'ajouter par la suite 2V supplémentaires, à savoir, la véracité et la valeur.

Le BIG DATA sous-entend une faible densité en information. En effet, avec la réduction des coûts des technologies de stockage, des données que leur faible densité en information rendait peu-utiles (e.g. logs Internet, historique des opérations, relevés de capteurs, etc.) sont stockées et analysées. Ceci s'explique par le besoin de plus en plus croissant des entreprises d'interroger toutes leurs sources de données plutôt qu'un échantillon afin de prendre en compte tous les faits. Comprendre et pouvoir extraire de la valeur sur ce très grand volume et cette grande variété de données en pleine croissance sont donc les défis majeurs auxquels sont confrontées les entreprises [7, 8, 9]. Ces dernières, animées par la concurrence ont besoin par exemple de connaitre leur e-réputation et de mesurer l'impact de leur stratégie de marketing, afin de mieux cibler par la suite leurs campagnes et adapter leurs produits aux besoins du client. Face à ces défis, les technologies du datamining permettent de traiter ces flux importants de données, autrement dit BIG DATA en vue d'en extraire de la connaissance, qui sera déterminante pour la prise de décisions. Par exemple, au cœur du processus de la connaissance client et donc de la gestion de la relation client, le datamining peut permettre d'extraire l'information utile à partir du BIG DATA jusqu'à la stratégie de marketing qui en découle.

Cependant cette grande disponibilité des données de nos jours, devient un problème dans l'analyse de données classique. En effet de nombreux algorithmes d'extraction des connaissances ont été rendus obsolètes face à de telles quantités de données [10, 11, 12]. Extraire des informations pertinentes de ces collections de données est devenu alors l'un des défis les plus importants et les plus complexes de la recherche en analyse de données. Des technologies plus efficaces sont à présent requises. Ces technologies permettraient de collecter, stocker, transmettre et traiter efficacement ces grands ensembles de données dans des intervalles de temps tolérables.

Un sujet sous-exploré mais non moins important est la grande dimensionnalité dans le BIG DATA [13, 14]. Ce phénomène, également connu sous le nom de « malédiction de la grande dimension », reflète l'explosion des attributs et l'impact combinatoire de nouveaux jeux de données volumineux avec des milliers, voire des millions d'attributs. En effet au cours des dernières années, la dimensionnalité des ensembles de données dans de nombreux domaines, comme la bio-informatique ou l'analyse de textes a considérablement augmenté. Ainsi, si nous analysons les ensembles de données publiés dans la populaire base de données **libSVM**, nous pouvons observer que la dimensionnalité maximale des données a augmenté à plus de 29 millions. Un autre problème est la myriade de types d'attribut et leurs combinaisons qui deviennent standard dans de nombreux scénarios du monde réel. Par exemple, le contenu multimédia sur Internet, qui représente environ 60% du trafic total [14, 15], est transmis dans des milliers de formats différents (audio, vidéo, images, etc.). Un autre exemple est le traitement du langage naturel, où plusieurs types de caractéristiques tels que des mots, des modèles n-grammes, etc., sont simultanément utilisés pour produire des modèles compréhensibles et fiables [16, 17]. Cependant de nombreuses méthodes d'apprentissage automatique (ML) ne peuvent pas traiter efficacement un aussi grand nombre d'attributs. De même, certains de ces algorithmes souffrent également lorsqu'ils sont confrontés à de grandes tailles d'échantillons (instances).

Pour résoudre ce problème, des techniques de réduction de la dimensionnalité peuvent être appliquées pour réduire le nombre d'attributs, de même que des méthodes de sélection d'instances afin de diminuer considérablement le nombre d'instances. Ce qui mènera à une amélioration des performances d'un processus d'apprentissage ultérieur. L'une des stratégies de réduction de dimensionnalité les plus

utilisées est la sélection d'attribut (**FS**), qui permet de réduire la dimensionnalité en supprimant les attributs non pertinents et redondants. En effet, tous les attributs d'un problème ne contribuent pas de manière égale aux modèles de prédiction et aux résultats. **FS** est donc nécessaire, maintenant plus que jamais, pour assurer un apprentissage et une prédiction rapide et rentable. Isoler des attributs de grande valeur d'un ensemble brut d'attributs potentiellement inutiles, redondants et bruités, tout en respectant les exigences de mesure et de stockage, est une tâche clé dans la recherche concernant les données volumineuses. Comme **FS** conserve les attributs d'origine, elle est particulièrement utile pour les applications où l'interprétation des modèles est importante.

Cependant, les méthodes **FS** existantes ne s'adaptent pas bien aux problèmes à grande échelle (en termes de nombre de caractéristiques et d'instances), parce que leur efficacité peut se détériorer considérablement. La grande dimensionnalité nécessite donc de nouvelles stratégies et méthodes **FS** pour traiter le problème d'explosion de caractéristiques. Dans les référentiels de données les plus connus comme UCI ou libSVM [1]), la dimensionnalité des ensembles de données nouvellement ajoutés atteint des tailles impressionnantes [18] (par exemple, près de 30 millions pour l'ensemble de données KDD2010).

Le traitement parallèle peut aider à résoudre ce problème, en permettant aux utilisateurs de travailler efficacement avec les BIG DATA [18, 19]. Des frameworks de programmation distribués efficaces, tels que MapReduce [20, 21] ont été proposés pour gérer le problème des données de grande taille.

Le framework MapReduce [20, 21] est apparu en 2003 comme un outil révolutionnaire pour gérer les BIG DATA. Il a été spécialement conçu par Google pour traiter des ensembles de données à grande échelle en distribuant automatiquement les données de manière distribuée sur des clusters (groupes d'ordinateurs). Le framework est en charge du partitionnement et de la gestion des données, de la récupération des erreurs, de la planification des tâches et de la communication, laissant aux programmeurs un outil transparent et évolutif pour exécuter facilement les tâches sur les systèmes distribués.

Apache Hadoop [20], [21] est une implémentation open-source de MapReduce pour un calcul fiable, évolutif et distribué. Bien qu'étant une implémentation open-source populaire de MapReduce, Hadoop n'est toutefois pas adapté à certaines applications, notamment l'informatique en ligne ou itérative, les paradigmes de

communication interprocessus élevés ou le calcul en mémoire [14, 21]. En effet, traditionnellement, les données sont placées dans les systèmes de stockage, puis sont rendues accessibles et placées en mémoire lors de l'analyse. Cela a pour effet de provoquer un goulot d'étranglement naturel qui réduit les performances. Même avec les disques SSD les plus rapides, il existera toujours un délai entre l'accès aux données, leur transfert en mémoire puis leur retour, pour que le prochain lot de données puisse être utilisé. Ainsi plus le volume de données à analyser augmente, plus le temps d'accès disque va augmenter et par conséquent celui du processus de datamining. Or la vitesse de l'analyse est très critique. Plutôt que d'analyser l'information pendant des jours ou des semaines, les entreprises souhaitent analyser en quelques minutes, passant au crible leurs opérations et les améliorant selon une situation donnée et présente, et non pas selon celle de la semaine dernière.

Le coût de la mémoire RAM ayant très fortement baissé, des moteurs de traitement de données massives permettant des calculs en mémoire et en parallèle sur un cluster ont vu le jour. C'est le cas d'Apache Spark qui a été inclus dans l'écosystème Hadoop [2, 3], comme un framework performant qui permet de calculer plus rapidement les données distribuées sur les BIG DATA. Il le fait en utilisant des primitives en mémoire qui lui permettent d'effectuer cent fois plus vite les traitements qu'Hadoop pour certaines applications. Le fait que cette plate-forme permette aux programmes utilisateurs de charger des données en mémoire et de faire des requêtes répétées signifie qu'elle est particulièrement bien adaptée au traitement en ligne et itératif (en particulier pour les algorithmes de Machine Learning). Ainsi grâce à Apache Spark, les technologies du Datamining profitent de vitesses de traitement élevées liées au mode de calcul en mémoire. Tout ceci entraine la refonte des algorithmes classiques. Ceci explique aussi notre choix porté sur Apache Spark dans les propositions faites dans cette thèse.

Cette thèse s'inscrit donc dans cette problématique d'adapter les algorithmes de datamining au contexte du Big data en s'appuyant sur l'environnement de calcul distribué. Nous étudions particulièrement la réduction de données dans le contexte du BIG DATA. Comment trouver des algorithmes de sélection d'attributs ou de sélection d'instances qui passent à l'échelle ?

2. Objectifs et contributions

Pour faire face à la faible densité de l'information dans le Big data, et aussi pour améliorer la vitesse de l'analyse, nous avons proposé dans le cadre de cette thèse de développer de nouveaux algorithmes de réduction de données (sélection d'attributs et la sélection d'instances) basés sur le calcul en mémoire et en parallèle sur un cluster. Pour cela, nous avons émis quatre propositions qui sont, respectivement :

- **PSF-mRMR** : Un algorithme à large échelle de sélection d'attributs qui combine deux critères de rang sur les attributs : (i) le mRMR (Minimum Redundancy Maximum Relevance) et (ii) le vecteur de poids d'un classifieur SVM. L'objectif de notre proposition est double, à savoir améliorer le pouvoir de prédiction d'un classifieur en proposant un nouveau critère de sélection d'attributs, et aussi profiter des performances de calcul en mémoire et en cluster de Spark pour améliorer la vitesse d'exécution des algorithmes de sélection d'attributs sur des données BIG DATA. Il existe quelques versions parallèles de mRMR, qui visent à réduire le goulot d'étranglement dans la version originale de mRMR. Celui-ci est lié au calcul de l'information mutuelle entre deux éléments : soit une caractéristique donnée avec la classe, soit une paire d'attributs en entrée. En effet, tous les attributs du jeu de données doivent être associés deux à deux afin de mesurer leur corrélation et choisir les attributs les moins corrélés aux autres et les plus pertinents. Ces méthodes utilisent une approche gloutonne afin de limiter le nombre de comparaisons entre attributs. Ainsi, au départ nous avons un sous-ensemble d'attributs à retourner qui est vide. Puis à chaque itération, l'attribut considéré comme le meilleur de l'ensemble par la méthode mRMR est ajouté à l'ensemble des attributs à retourner. Le processus s'arrête une fois que le nombre d'attributs à retourner est obtenu. Donc l'algorithme itère autant de fois qu'il y a d'attributs à retourner. Par ailleurs, ces algorithmes effectuent plusieurs jobs or dans un environnement massivement distribué, la performance globale d'un processus est améliorée quand on peut minimiser le nombre de « jobs » (les « aller/retours » entre les machines distribuées). Cela impacte le temps d'exécution, mais aussi le transfert de données. Dans le cas de la réduction de données, trouver les meilleurs attributs en un seul job simplifié serait donc préférable. Aussi, nous proposons une méthode qui, en

une seule itération, choisit les meilleurs attributs du jeu de données d'entrée et les retourne.

- **PS_SVM-RFE** : Une version à large échelle (parallèle et évolutive) d'une méthode de sélection de caractéristiques bien connue appelée RFE-SVM. Les résultats expérimentaux ont démontré l'évolutivité de PS_SVM-RFE et ses performances en termes de réduction du temps d'exécution en comparaison avec la méthode RFE-SVM classique.
- **PNFS_Spark** : Un algorithme à large échelle de sélection de variables qui introduit un nouveau critère de rang sur les variables basées sur l'utilisation de la valeur médiane de la pertinence des attributs avec l'étiquette de classe. Dans cette méthode, la relevance de chaque attribut est d'abord calculée et la valeur médiane est déterminée. Les voisins de chaque attribut sont déterminés en se basant sur l'information mutuelle entre les attributs. On considère un attribut comme voisin d'un autre si l'information mutuelle entre les 2 est supérieure à la valeur médiane. Le score d'un attribut est alors sa relevance multipliée par son nombre de voisins.
- **LSIS** : le principal problème des méthodes de sélection d'instance parallèle est que la plupart d'entre elles sont conçues pour le classifieur des k-plus-proches-voisins. Ainsi les instances sélectionnées par ces algorithmes ne sont souvent adaptées qu'à ce type de classifieur. **LSIS** est une toute nouvelle méthode qui ne dépend pas du classifieur des k-plus-proches-voisins et qui peut être appliquée à tout type de classifieur. Elle a présenté de très bonnes performances en termes de précision de classification et a montré de bons résultats tant en terme de temps d'exécution que de passage à l'échelle (scalabilité) comparée aux algorithmes de sélection d'instance de la littérature.

3. Organisation de la thèse

L'ensemble de nos travaux et résultats sont résumés dans cette thèse, qui comprend **cinq chapitres**, outre une introduction et une conclusion.

– Dans le **Chapitre 1**, nous avons fait un état de l'art sur les technologies de traitement des données Big data. Nous y avons présenté le paradigme de programmation MapReduce, Apache Hadoop, Apache Spark.

– Le **Chapitre 2**, aborde les concepts et les principes de base de la sélection d'attributs et de la sélection d'instances.

– Le **Chapitre 3** fait un survol des algorithmes de réduction de données (sélection d'attributs et sélection de variables) à large échelle.

– Les principes ainsi que les fonctionnalités de nos propositions sur la sélection d'attributs à large échelle sont détaillés dans le **Chapitre 4**. Nous proposons notamment trois algorithmes : PSF-mRMR, PNFS_Spark et PS-RFE-SVM.

– Dans le **Chapitre 5**, nous donnons les détails de notre proposition relative à la sélection d'instances à large échelle, qui a pour objectif de sélectionner un sous ensemble d'instances qui comporte le plus d'informations pour un objectif de classification.

La thèse se termine par une synthèse de nos travaux et de nos différentes contributions, et liste quelques perspectives de recherche, à partir des résultats que nous avons obtenus.

Première partie
Etat de l'art

Chapitre I

Le framework Map Reduce

Le problème majeur de nos jours est que la taille des ensembles de données augmente de plus en plus aussi bien verticalement et qu'horizontalement. Avec cette croissance rapide des données, les algorithmes de sélection d'attributs ou d'instances classiques ne fonctionnent pas correctement car ils ont été conçus pour le calcul centralisé [14]. Ainsi il devient urgent d'adapter ces algorithmes au contexte actuel de croissance rapide des données. Pour faire face à cette problématique, les techniques de calcul distribué telles que MapReduce [11, 12] et sa mise en œuvre open-source Apache Hadoop peuvent être une solution. Mais la programmation parallèle MapReduce avec Apache Hadoop n'est pas très adaptée aux méthodes de réduction de données qui procèdent généralement de manière itérative [12]. C'est pourquoi, un framework plus adapté à ce genre d'algorithmes a été présenté comme une alternative à Apache Hadoop [20]. Il s'agit du framework Apache Spark.

Comme la mise en œuvre des algorithmes nécessite des connaissances sur des systèmes de données parallèles, dans ce chapitre, nous discutons d'abord des systèmes parallèles existants tels que le modèle de programmation MapReduce dont nous détaillons le principe de fonctionnement et l'architecture de base. Ensuite nous présentons l'implémentation open source Apache Hadoop de MapReduce et ses limites. Nous finissons par la présentation de Spark et une comparaison entre les deux framework.

1. Modèle de Système de traitement de données parallèle

Le traitement parallèle divise les grands problèmes en problèmes plus petits et effectue l'exécution ou le calcul des processus simultanément (en même temps) [7].

Les systèmes de traitement de données parallèles utilisent une architecture "shared nothing" [7] où plusieurs machines indépendantes sont interconnectées dans un réseau et chaque nœud ne stocke généralement qu'une partie des données globales. Dans un tel environnement, le meilleur moyen de réaliser le parallélisme des données est d'amener le calcul aux données. Un exemple courant est le problème WordCount, pour compter le nombre de mots au total. Ce problème permet à la plupart du travail de calcul d'être effectué localement (appelé localité de données). Chaque machine (ouvrier) envoie alors un seul nombre à une seule machine (maître) pour calculer la somme finale.

Les systèmes parallèles sont utilisés pour augmenter l'accélération et mesurer l'évolutivité. Le Speedup décrit la capacité à résoudre un même problème avec une taille d'entrée fixe plus rapidement, généralement en ajoutant plus de nœuds ou en utilisant une forme de parallélisme. L'évolutivité décrit la capacité à résoudre des problèmes plus importants en ajoutant plus de nœuds. Ceci est également connu sous le nom d'extensibilité horizontale, ou scaleout. En revanche, il existe également une évolutivité verticale (scaleup), où un matériel plus puissant est utilisé au lieu d'ajouter plus de nœuds.

L'accélération linéaire (linear speedup) signifie qu'un problème peut être résolu deux fois plus vite en doublant les ressources de calcul. L'échelonnement linéaire (linear scaleout) signifie qu'un problème dont la taille est doublée peut être résolu en même temps en utilisant deux fois plus de ressources de calcul.

Dans les sections suivantes, nous discutons de plusieurs systèmes parallèles de traitement de données pertinents pour l’implémentation de la sélection d’attributs.

1.1 Comparaison entre systèmes parallèles

Un modèle de systèmes parallèles réputé est le Message Passing Interface (MPI) qui est un protocole de communication de transfert de messages portable normalisé pour le calcul parallèle et qui prend en charge les communications point à point et collective [7, 5]. La norme définit la syntaxe, le protocole et la sémantique d'un noyau de bibliothèques pour permettre l'écriture de programmes de transmission de messages portables en C, C ++ et Fortran. Les objectifs MPI sont la haute performance, l'évolutivité et la portabilité. Les systèmes MPI activent le modèle de programmation parallèle de transmission de message, dans lequel les données sont déplacées de l'espace d'adressage d'un processus à celui d'un autre processus via des opérations sur chaque processus. L'idée principale pour MPI est d'effectuer toutes les parties d'une tâche en parallèle sans aucune communication entre les processus [7, 5]. MPI ne convient pas au niveau de parallélisme à petit grain, par exemple, pour exploiter le parallélisme des plates-formes multi-core pour le multitraitement à mémoire partagée [5]. OpenMP est une interface de programmation d'application (API) qui prend en charge la programmation multiprocesseur à mémoire partagée multiplateforme [7]. OpenMP est devenu le standard pour la programmation parallèle en mémoire partagée, mais il n'était toujours pas adapté aux systèmes de mémoire distribuée jusqu'à ce que Coarray Fortran (CAF) [7] soit proposé. CAF est une extension Fortran qui repose sur la norme MPI pour le traitement parallèle sur les systèmes de mémoire distribués. CAF permet deux niveaux de granularité : le parallélisme de petit grain avec OpenMP et le large parallélisme des grains avec CAF à base de MPI [7, 5].

MPI utilise un modèle de programmation de mémoire distribuée qui s’appuie sur un parallélisme explicite. Ce qui signifie que le développeur doit traiter explicitement de nombreux problèmes, comme l'assignation de tâches à des travailleurs répartis sur un nombre potentiellement important de machines, la synchronisation entre les différents travailleurs, le partage de résultats partiels d'un travailleur requis par un autre, etc. OpenMP utilise le parallélisme de la mémoire partagée, ce qui signifie que le développeur doit coordonner explicitement l'accès aux structures de données partagées via des primitives de synchronisation, gérer explicitement la synchronisation des processus par des dispositifs et résoudre des problèmes communs

tels que les interblocages et les conditions de concurrence. Les développeurs doivent garder une trace de la façon dont les ressources sont mises à la disposition des travailleurs et doivent accorder une grande attention aux détails du système de bas niveau. En outre, MPI et OpenMP sont conçus pour résoudre les problèmes liés aux processeurs et ont un support limité pour traiter de très grandes quantités de données d'entrée.

2. Modèle de programmation MapReduce

MapReduce [3] est un paradigme de programmation et un environnement d'exécution proposé par Google en 2003. Il permet la parallélisation de calculs manipulant de gros volumes de données dans des clusters de machines (centaines, milliers de CPU) d'une manière évolutive, robuste et efficace.

L'idée de MapReduce est qu'au lieu de déplacer de grandes quantités de données, il est beaucoup plus efficace dans la mesure du possible de déplacer le code vers les données. MapReduce répartit les données sur les disques locaux des nœuds d'un cluster et exécute les processus sur les nœuds qui contiennent les données. La tâche complexe de gestion du stockage dans un tel environnement de traitement est généralement gérée par un système de fichiers distribué situé sous MapReduce. Ce système de gestion de fichiers est appelé HDFS.

Fondamentalement, un job MapReduce commence par recevoir en entrée des données stockées sur le système de fichiers distribué sous-jacent. MapReduce nécessite un développeur pour définir un mappeur et une fonction réducteur. Il faut alors définir une fonction map et une fonction reduce. La fonction map est appliqué à chaque entrée (divisée sur un nombre arbitraire de fichiers) pour générer un certain nombre de paires clé/valeur intermédiaires. La fonction reduce est appliquée à toutes les valeurs associées à la même clé intermédiaire pour générer des paires clé/valeur en sortie.

Il fournit une abstraction simple pour le développeur en lui cachant de nombreux détails du système et en les gérant de manière transparente. C'est l'un des avantages les plus significatifs de MapReduce car un développeur peut se concentrer sur ce que les calculs doivent effectuer, sans avoir à se soucier de comment ces calculs sont réellement effectués [3]. De plus ce modèle gère les éventuelles fautes (pannes) et les communications entre les machines. Un autre avantage de MapReduce est qu'il

n’implique aucune modification des structures de données (flux de données implicite) et l’ordre des opérations n’a pas d’importance dans l’environnement d’exécution.

MapReduce est intégré dans des environnements open-source tels qu’Apache Hadoop ou Apache Spark.

3. Implémentation de MapReduce

Ici nous allons présenter les 2 implémentations open source de MapReduce les plus connus : Apache Hadoop et Apache Spark.

3.1 Hadoop

Hadoop est une plateforme logicielle open-source basée sur le modèle MapReduce et qui permet de stocker des données et de lancer des applications sur des grappes de machines standard. Cette solution offre un espace de stockage massif pour tous les types de données, une immense puissance de traitement et la possibilité de prendre en charge une quantité de tâches virtuellement illimitée.

HDFS (Hadoop Distributed File System) est un système de fichier distribué permettant de stocker et de récupérer des fichiers en un temps record. Il s’agit de l’un des composants basiques du framework Apache Hadoop, et plus précisément de son système de stockage. Chaque fichier copié dans HDFS est divisé en blocs, la taille de bloc par défaut est de 128 Mo, et chaque bloc est stocké sur quelques nœuds. Le facteur de réplication définit le nombre de nœuds envoyés : une valeur élevée augmente la tolérance aux pannes et parfois la localisation des données, mais ralentit le chargement des données et nécessite plus d'espace.

YARN est le gestionnaire de cluster de l'écosystème Apache Hadoop, qui possède toutes les ressources (nœuds) d'un cluster et gère toutes les applications s'exécutant sur le cluster. Les applications écrites pour YARN ont leurs propres tâches de maître et de travailleur, mais YARN recevra et ordonnera les soumissions de travaux et accordera les ressources [4].

Hadoop est basé sur le principe master / slave (maître / esclave). L’architecture du framework est composée de nœuds maîtres qui cordonnent de nombreux nœuds esclaves. C’est ce principe qui est utilisé dans la construction de HDFS, qui est basé sur un NameNode et divers DataNodes subordonnés. Le NameNode gère plusieurs

métadonnées du système de fichiers, de la structure des répertoires et des DataNodes subordonnés. Pour minimiser la perte de données, les fichiers sont découpés en différents blocs et stockés sur différents nœuds.

3.2 Spark

Spark est un framework de calcul itératif basé sur la mémoire qui utilise des techniques in-memory de traitements de données permettant de stocker l'ensemble du processus en mémoire, le jeu de données initial et les résultats intermédiaires [5, 6, 7].

L'abstraction de base de Spark est le RDD pour Resilient Distributed Dataset. C'est une structure de donnée immutable qui représente un Graph Acyclique Direct des différentes opérations à appliquer aux données chargées par Spark.

Un RDD peut résider en mémoire, disque ou en combinaison. Les éléments RDD sont automatiquement partitionnés dans le cluster. Puisque les RDD peuvent être gardés en mémoire, les algorithmes peuvent parcourir plusieurs fois les données RDD de manière très efficace.

Un RDD est immuable. Une fois créé, il ne peut être modifié. Un RDD peut reconstruire automatiquement toutes les données perdues lors des pannes. En effet Spark récupère les partitions perdues en les recalculant à partir des données de base [8].

Un calcul distribué avec Spark commence toujours par un chargement de données via un Base RDD.

Le RDD correspond en fait à une sorte de plan d'exécution contenant toutes les informations des opérations qui vont s'appliquer sur les données.

Chaque RDD peut être : soit une collection stockée dans un système de stockage externe, tel qu'un fichier dans HDFS, soit un ensemble de données dérivé créé en appliquant des opérateurs (par exemple map, group by, reduce, hash join) à d'autres RDD. Les opérateurs appliqués aux éléments de données peuvent : soit produire d'autres RDD (transformations), soit renvoyer des valeurs (actions) [9]. Cependant, ces opérations sont calculées en utilisant la stratégie d'évaluation paresseuse afin d'effectuer un calcul minimal et d'empêcher l'utilisation de la mémoire inutile. Les RDD ne sont pas mis en cache dans la mémoire par défaut. Par conséquent, lorsque les données sont réutilisées, une méthode ***persist*** est nécessaire pour éviter le recalcul.

Apache Spark utilise l'architecture master / worker et possède un gestionnaire de clusters qui fonctionne correctement. Spark peut être en mode standalone , ou intégré avec Apache Mesos et Hadoop YARN [4].

Un cluster Spark a un seul coordinateur appelé master et plusieurs nœuds esclaves appelés workers. Le programme pilote qui s'exécute sur le nœud maître du cluster Spark planifie l'exécution du travail et négocie avec le gestionnaire de cluster. Le pilote exécute la fonction **main ()** de l'application et accède au cluster via un objet appelé **SparkContext**. SparkContext connaît le gestionnaire de ressources utilisé et peut lui demander des ressources sur le cluster.

Le worker exécute une instance Spark où est créé un agent distribué responsable de l'exécution des tâches appelées executor (exécuteur). Executor exécute plusieurs tâches individuelles dans un job Spark donné.

Le pilote et les exécuteurs peuvent être exécutés sur le même groupe de clusters Spark de manière horizontale ou sur des machines séparées, c'est-à-dire d'une manière verticale dans un groupe de cluster Spark ou dans une configuration de machines mixte.

La figure suivante montre l'architecture d'un cluster Spark [4].

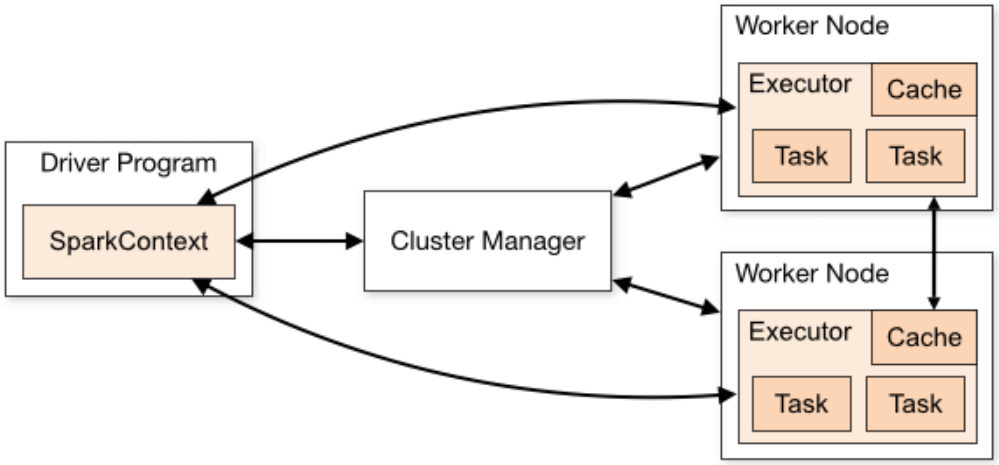

Figure I.1 - Architecture d'un cluster Spark.

3.3 Etude comparative entre Hadoop et Spark

Pour les algorithmes qui fonctionnent de manière itérative, c'est à dire qui effectuent plusieurs tours de calcul sur les mêmes données, Hadoop ne convient pas parce qu'il manque de support intégré pour le processus itératif. En effet, Apache Hadoop utilise le modèle de programmation MapReduce qui lit et écrit à partir du disque, ce qui ralentit la vitesse de traitement. Ainsi, Apache Spark a été présentée comme une alternative à Hadoop pour surmonter ces problèmes. Spark est un framework de calcul itératif basé sur la mémoire permettant de disposer d'un surcroît de puissance via l'utilisation de techniques in-memory de traitements de données permettant de stocker l'ensemble du processus en mémoire. Il réduit ainsi le nombre de cycles de lecture / écriture sur le disque. C'est pour cela qu'il convient aux applications utilisant des opérations itératives. C'est donc un framework de calcul en mémoire parallèle très rapide spécialisé dans le calcul de grands ensembles de données distribuées.

Les expériences de [10, 8] ont montré que Spark surpasse de deux ordres de grandeur les travaux classiques de MapReduce en termes de vitesse.

4. Conclusion

Dans ce chapitre, nous avons présenté le paradigme de programmation MapReduce qui permet la parallélisation de calculs manipulant de gros volumes de données d'une manière évolutive, robuste et efficace. Nous avons ensuite présenté les différents framework telles qu'Apache Hadoop, qui implémente MapReduce. Mais Apache Hadoop n'est pas très adaptée à des méthodes telles que les techniques de réduction de données qui procèdent généralement de manière itérative. Ainsi, le framework Apache Spark a été présenté comme une alternative à Hadoop et est plus adapté aux algorithmes qui fonctionnent de manière itérative.

Dans le chapitre qui suit, nous allons présenter les techniques de réduction de données telles que la sélection d'attributs et la sélection d'instances et les objectifs qu'elles se fixent.

Chapitre II

Réduction de données

De nos jours dans beaucoup de domaines tels que la fouille de données, le traitement des images, l'analyse de données en bio-informatique, l'apprentissage automatique, les applications nécessitent de traiter des données décrites par un très grand nombre d'attributs et d'instances [1, 11]. Par exemple une image peut être décrite par des millions de pixels. En bio-informatique, les données manipulées peuvent engendrer des niveaux d'expression de plusieurs milliers de gènes.

Il est alors très fréquent de trouver dans ces grands ensembles de données des attributs ou instances corrélés et qui expriment des informations similaires [12, 13]. Ces attributs sont dits redondants. Il est également possible de trouver des attributs ou instances non pertinents c'est à dire non utiles au problème étudié [12, 13]. De plus, ces données redondantes ou non pertinentes altèrent la performance des applications. Dans le cas de l'apprentissage supervisé par exemple, la précision de la classification peut se détériorer ou le temps d'apprentissage de la classification peut devenir très élevé. C'est pour cela que la sélection d'attributs et la sélection d'instances jouent un rôle important dans l'exploration de données puisqu'elles se donnent comme objectif de supprimer les caractéristiques ou instances non pertinentes et redondantes des données d'origine [1, 13]. Leur objectif est de réduire le temps de calcul, d'améliorer les performances de prédiction, de construire des modèles plus simples et plus complets et de permettre une meilleure compréhension des données dans les problèmes d'apprentissage automatique ou d'exploration de données.

Dans ce chapitre nous allons présenter les techniques de réduction de données telles que la sélection d'attributs et la sélection d'instances, nous étudierons leur fonctionnement et les principales méthodes qui existent dans la littérature.

1. Sélection d'attributs

La sélection d'attributs est généralement définie comme un processus de recherche permettant de chercher dans un ensemble de grande taille un sous-ensemble d'attributs "pertinent" tout en évitant les attributs redondants [22, 23, 24, 25]. La pertinence des attributs dépend du problème étudié. En général, le problème de sélection de caractéristiques peut être défini par :

Définition II.1 : Soit F un ensemble d'attributs donné F = $\{f_1,.., f_n\}$. Une instance V est représentée par un vecteur m-dimensionnel ($v_1,.., v_n$), où v_j est la valeur de la caractéristique f_j dans V. Soit O (F', T) la fonction objective qui évalue le sous-ensemble F' de F en utilisant les données T. L'objectif est de trouver le meilleur sous-ensemble, c'est à dire celui qui maximise la fonction d'évaluation. Un sous-ensemble F_1 est meilleur que F_2 si O (F_1, T)> O (F_2, T).

Pour faire de la sélection d'attributs il est possible de procéder de trois manières différentes qui sont les suivantes [25, 26] :

- Fixer la taille du sous-ensemble d'attributs à sélectionner et chercher le meilleur sous-ensemble de cette taille.
- Fixer un seuil et sélectionner le plus petit sous-ensemble dont la performance est plus grande ou égale à ce seuil. Par exemple en classification supervisée on pourrait fixer une précision p et chercher le plus petit sous ensemble qui donne une précision au minimum égale à p pour le classifieur.
- Sélectionner le sous-ensemble qui optimise à la fois les performances et réduit la taille du sous ensemble.

Bien que la sélection d'attributs puisse concerner plusieurs tâches de fouille de données, nous nous intéresserons particulièrement dans cette thèse à la sélection d'attributs pour la classification supervisée. Dans ce contexte la sélection d'attributs a les objectifs suivants [27] :

- Trouver le plus petit sous ensemble qui améliore la précision du classifieur une fois les attributs sources de bruit enlevés.
- Réduire le temps d'apprentissage et le rendre ainsi moins couteux.
- Permettre une meilleure compréhension du phénomène étudié.

- Permettre une meilleure généralisation des données en évitant le sur-apprentissage.

En présence de centaines voire de milliers d'attributs, il y a de fortes chances que des attributs soient corrélés et expriment des informations similaires. On dit qu'ils sont redondants. Les attributs qui fournissent le plus d'informations pour la classification sont quant à eux appelés attributs pertinents. La sélection d'attributs doit permettre de trouver un sous-ensemble optimal d'attributs constitué. De plus, cet ensemble doit permettre de satisfaire au mieux l'objectif fixé, à savoir la précision et la rapidité de l'apprentissage ou bien encore l'explicabilité du classifieur proposé [28].

Il est possible de regrouper les techniques de sélection d'attributs en fonction de l'objectif visé [29, 30]. On a alors quatre classes distinctes :

- « **Classic** ». Cette classe considère la sélection d'attributs comme un processus permettant de réduire l'ensemble des attributs d'une taille N à une taille M, avec M < N. Cette réduction de la dimensionnalité permet une accélération de la vitesse du traitement, une augmentation et une amélioration de la précision du traitement ;

- « **Idealized** ». Ici la sélection d'attributs est vue comme un processus qui permet de trouver le sous-ensemble de taille minimale nécessaire et suffisant pour atteindre l'objectif fixé ;

- « **Improving prediction accuracy** ». La sélection d'attributs est un processus qui permet de choisir un sous-ensemble d'attributs afin d'améliorer la précision de la prédiction ou diminuer la taille de la structure sans diminution significative de la précision de prédiction du classifieur construit en utilisant seulement les variables sélectionnées.

- « **Approximating original class distribution** ». La sélection d'attributs est un processus qui permet de sélectionner un sous-ensemble de variables, tel que la distribution des classes résultantes soit aussi proche que possible de la distribution des classes étant donné l'ensemble complet des variables.

L'exemple suivant [31, 32] explique cette problématique :

Exemple II.1 : Considérons un problème de classification, avec 5 descripteurs représentés par attributs booléens F1, . . ., F5. Soit C = g (F1, F2) la fonction de classification, où g est une fonction booléenne. Supposons qu'il existe les relations suivantes entre les attributs : F2 = F3 et F4 = F5 (notre problème concerne donc huit

exemples possibles). F1 est indispensable afin de déterminer la fonction cible. Un des attributs F2 ou F3 peut être enlevé car F2 = F3, C peut également s'écrire sous la forme g (F1, F3); mais un des attributs F2 ou F3 doit être utilisé pour définir C. Par contre, F4 et F5 peuvent être éliminés car ils ne sont pas utiles pour définir C. Pour ce problème deux sous-ensembles d'attributs optimaux peuvent donc être sélectionnés : {F1, F2} ou {F1, F3}.

Cet exemple montre clairement que les notions de pertinence et redondance jouent un rôle fondamental dans la sélection d'attributs et c'est pour cela que nous allons les définir dans la section suivante.

1.1 Pertinence et redondance d'attributs

1.1.1 Pertinence d'un attribut

La performance d'un algorithme d'apprentissage dépend fortement des attributs utilisés dans la tâche d'apprentissage. La présence d'attributs redondants ou non pertinents peut réduire cette performance. Il existe plusieurs définitions de la pertinence d'un attribut dont l'une des plus connue est celle qui considère un attribut comme étant soit très pertinent, soit peu pertinent, soit non pertinent [33, 34].

- **Très pertinent :** Un attribut est dit très pertinent si son absence peut conduire à une dégradation non négligeable de la performance du système de classification utilisé.
- **Faiblement pertinent** : Un attribut est dit faiblement pertinent s'il n'est pas "très pertinent" et s'il existe un sous-ensemble F tel que la performance de $F \cup \{f_i\}$ soit significativement meilleure que la performance de V. Ainsi, un attribut faiblement pertinent ou peu pertinent n'est pas toujours important mais peut le devenir pour un sous ensemble optimal dans certaines conditions.
- **Non pertinent :** Les attributs qui ne sont ni "peu pertinents" ni "très pertinents" représentent les attributs non pertinents. Cela indique donc que ces attributs ne sont pas du tout nécessaires dans un sous ensemble d'attributs optimal. Ces attributs seront en général supprimés de l'ensemble d'attributs de départ.

Un sous ensemble optimal ne devrait inclure que les attributs très pertinents, aucun attribut non pertinent et un sous ensemble d'attributs faiblement pertinents.

1.1.2 Redondance d'un attribut

La notion de redondance d'attributs se comprend intuitivement et elle est généralement exprimée en terme de corrélation entre attributs. On peut dire que deux attributs sont redondants entre eux si leurs valeurs sont complétement corrélées [35, 36].

1.2 Le cadre général d'un algorithme de sélection d'attributs

Une méthode de sélection passe généralement par quatre étapes [37, 38].
Les deux premières consistent à initialiser le point de départ à partir duquel la recherche va commencer et à définir une procédure de recherche ou une procédure de génération de sous-ensembles de caractéristiques. Une fois la stratégie de recherche définie, et les sous-ensembles générés, une méthode d'évaluation est définie dans la troisième étape. On a donc :

- **Un point de départ** : point à partir duquel la recherche commence.
- **Une procédure de recherche** : qui permet de générer l'espace des sous-ensembles candidats et de le parcourir.
- **Une fonction d'évaluation** : qui permet de mesurer la capacité de la variable, ou l'ensemble des variables, à différencier les classes. Autrement dit, elle donne une mesure de la pertinence de la variable dans un problème de classification ou de régression.
- **Un critère d'arrêt** : qui permet de décider si on continue la recherche du sous-ensemble optimal ou on arrête le processus.
- **Un processus de validation** : pour vérifier, si l'objectif souhaité est atteint.

Les étapes 2 et 3 se répètent jusqu'au critère d'arrêt. Les 5 étapes sont détaillées dans les sections suivantes.

1.2.1 Initialisation et procédures de recherche

La première des choses à faire avant d'appliquer la procédure de recherche est de déterminer sur quel point de l'espace de caractéristiques la recherche va commencer [39, 36]. Il s'agit du point de départ (ou direction de recherche). Par exemple, une recherche peut commencer par un ensemble vide d'attributs, on ajoute successivement des attributs à chaque itération. Inversement, la recherche peut commencer avec l'ensemble de tous les attributs et à chaque itération la caractéristique la moins pertinente est éliminée. Il est possible aussi de commencer la recherche par un sous-ensemble quelconque d'attributs.

Après le choix du point de départ, une procédure de recherche servant à générer des sous-ensembles d'attributs doit être définie. Les stratégies de recherche peuvent être rangées en trois catégories : exhaustive, heuristique et aléatoire.

a) recherche exhaustive

Dans cette stratégie, une recherche exhaustive est faite sur tous les sous-ensembles d'attributs dans le but de choisir le "meilleur" sous-ensemble d'attributs. Cette approche garantit de trouver le sous-ensemble optimal. Mais le problème majeur de cette approche est que le nombre de combinaisons augmente de manière exponentielle en fonction du nombre d'attributs. Quand la taille des attributs N devient grande, les 2^N combinaisons possibles rendent la recherche exhaustive impossible [36, 40].

b) recherche heuristique

Dans cette stratégie, une approche heuristique pour guider la recherche est utilisée. Les algorithmes qui utilisent cette approche sont en général itératifs et rajoutent ou éliminent des attributs à chaque itération. Ils ne garantissent pas de résultats optimaux parce qu'ils ne parcourent pas totalement l'espace de recherche mais ont comme avantage d'être simple et rapide. Les trois sous-catégories les plus connues de cette approche sont :

Forward : cette approche est également appelée ascendante, elle part d'un ensemble d'attributs vide auquel sont ajoutées à chaque itération une ou plusieurs

caractéristiques.

Backward : c'est une approche inverse de "Forward". Elle est aussi appelée approche descendante. L'ensemble de départ représente l'ensemble total des attributs et à chaque itération, un ou plusieurs attributs seront supprimées.

Stepwise (bidirectionnelle) : cette approche est un mélange des deux précédentes. Elle consiste à ajouter ou supprimer des attributs au sous-ensemble courant.

c) recherche aléatoire (stochastique ou non-déterministe)

Pour un ensemble donné et une initialisation particulière, une stratégie de recherche heuristique retourne toujours le même sous-ensemble, ce qui la rend très sensible au changement de l'ensemble de données. Dans cette stratégie, les sous-ensembles d'attributs sont générés d'une manière totalement aléatoire dans le but de trouver le meilleur sous ensemble d'attributs. Ce qui signifie que le sous-ensemble courant ne provient pas d'une augmentation ou diminution des attributs du sous-ensemble précédent. En outre, les stratégies de recherche aléatoire convergent en général rapidement vers une solution "semi-optimale". L'obtention de bons résultats avec cette méthode nécessite un choix judicieux d'un certain nombre de paramètres.

Le tableau suivant fait une comparaison entre les méthodes de recherche et montre les avantages et inconvénients de chaque méthode.

	Rigueur (Exactitude)	**Complexité**	**Avantages**	**Inconvénients**
Exhaustive	Trouve toujours la solution optimale	Exponentielle	Grande exactitude	Complexité très élevée
Séquentielle	Bonne mais non optimale	Quadratique $O(2^n)$	Simple et rapide	Pas de retour arrière
Aléatoire	Bonne avec un bon contrôle de para- mètres	Généralement basse	Désigné pour échapper des optima locaux	Difficile de régler les paramètres

Tableau II.1 – Comparaison entre méthodes de recherche [36]

1.2.2 Stratégies d'évaluation

Il existe 2 grandes stratégies d'évaluation pour la sélection d'attributs suivant que l'on évalue les attributs individuellement ou des sous-ensembles d'attributs [36] :

L'évaluation individuelle. Dans cette approche, chaque attribut est évalué indépendamment des autres, à partir des seules informations fournies par les données et un poids (score) lui est accordée selon son degré de pertinence. On obtient alors une liste des attributs ordonnés selon leur poids. On sélectionnera ensuite les m premiers attributs. Cela garantit ainsi d'avoir des attributs pertinents, mais cette approche ne peut supprimer les attributs redondants car ces attributs ont probablement des scores similaires. Pour des données de grande dimension contenant un grand nombre d'attributs corrélés, cette technique entraine des résultats éloignés des optimaux, car elle ne gère pas du tout la redondance. Le principal intérêt de cette méthode est sa complexité linéaire, qui lui permet de traiter des données de grande dimension. Ainsi cette approche peut être utilisée dans un premier temps pour réduire la taille de l'ensemble de départ.

L'évaluation d'un sous-ensemble. Il existe un grand nombre de techniques de sélection qui se basent sur l'évaluation des sous-ensembles pour gérer à la fois la redondance et la pertinence. Dans le processus de sélection, la génération de sous-ensembles produit des sous-ensembles candidats suivant une stratégie de recherche. Chaque sous-ensemble d'attributs candidat est évaluée par une certaine mesure d'évaluation et comparé avec le meilleur sous-ensemble d'attributs obtenu précédemment par rapport à cette mesure. Si le sous-ensemble courant est meilleur, il remplace le meilleur sous-ensemble d'attributs précédent. Le processus de génération et d'évaluation d'un sous-ensemble est répété jusqu'à ce qu'un critère d'arrêt soit satisfait. L'avantage est que les mesures d'évaluation utilisées par cette approche prennent en compte l'existence et l'effet d'attributs contrairement aux méthodes d'évaluation individuelle.

1.2.3 Critère d'arrêt

Certains critères doivent être définis pour arrêter le processus de recherche sur les sous-ensembles de caractéristiques. Les critères d'arrêt les plus fréquents sont [36, 43, 44] :

1. Un certain seuil est atteint, ce seuil peut être un nombre spécifique (comme par exemple un nombre minimum d'attributs ou un nombre maximal d'itérations) ;

2. L'addition (ou suppression) de n'importe quel attribut ne produit pas un sous-ensemble meilleur ;

3. Un sous-ensemble sélectionné est suffisamment bon (par exemple un sous-ensemble peut être suffisamment bon si son taux d'erreurs lors de la classification est inférieur au taux d'erreurs admissible pour une tâche donnée).

1.2.4 Procédure de validation

Une manière de faire la validation des résultats est de mesurer directement ces derniers en utilisant des connaissances a priori sur les données [36, 45]. Par exemple, si les attributs pertinents sont connus à l'avance, nous pouvons comparer les attributs sélectionnés avec cet ensemble d'attributs connu (les connaissances sur les attributs non pertinents ou redondants peuvent aussi aider). Toutefois, dans les applications du monde réel, généralement on ne dispose pas de connaissances a priori. Ainsi nous devons utiliser des méthodes indirectes, en surveillant le changement des performances par rapport aux changements des attributs. Par exemple, si nous utilisons le taux d'erreur de classification comme un indicateur de performance pour le traitement, pour un sous-ensemble d'attributs sélectionné, nous pouvons simplement suivre l'expérience « avant et après »pour comparer le taux d'erreur du classifieur sur l'ensemble complet d'attributs et sur le sous-ensemble d'attributs obtenu.

1.2.5 Les approches pour la sélection d'attributs

Il est à ce jour impossible de recenser toutes les méthodes de sélection d'attributs existantes dans la littérature. Face à ce grand nombre, plusieurs auteurs se sont intéressés à leur caractérisation en grandes familles de méthodes. Selon ces auteurs, les méthodes de sélection d'attributs peuvent être classées en trois catégories principales : ”filter”, ”wrapper” et ”embedded” [36, 46, 47, 48].

1.2.5.1 Les méthodes de type filtre (filter method)

Le modèle ”filter” a été le premier utilisé pour la sélection de caractéristiques. Dans les méthodes « filtres », les caractéristiques sont sélectionnées sur la base de leurs scores dans divers tests statistiques pour leur pertinence ou leurs pouvoirs discriminants avec la variable de résultat [36, 48, 49]. Les méthodes qui se basent sur ce modèle pour l'évaluation des caractéristiques, utilisent souvent une approche heuristique comme stratégie de recherche. La procédure de sélection de sous-ensembles est indépendante de l'algorithme d'apprentissage et est généralement une étape de prétraitement. Les méthodes « filtres » ne tiennent pas compte des interactions entre attributs, ni de la performance du classifieur. Les méthodes « filtres » offrent une meilleure complexité de calcul pour les ensembles de données ayant un grand nombre de caractéristiques.

La procédure du modèle ”filter” est illustrée [36, 48, 49] par la figure (I.2).

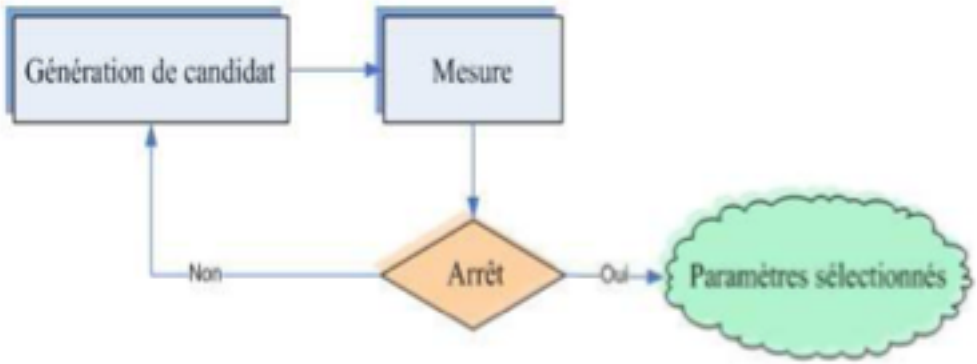

Figure I.2 - Procédure du modèle « filtre » [36]

Les caractéristiques sont généralement évaluées par des mesures calculées pour chacune des caractéristiques. Soit $X = \{x_k | \, x_k = (x_{k1}, x_{k2}, \ldots, x_{kn}),\ k = 1,\ 2, \ldots,\ m\}$ un ensemble de m exemples d'apprentissage dans un espace de représentation comportant n attributs. Soit $Y = \{y_k,\ k = 1,\ 2, \ldots,\ m\}$ où y_k représente l'étiquette de la classe de l'exemple x_k. $x^i = (x_{1i},\ x_{2i},\ , x_{mi})$ représente la $i^{ème}$ caractéristique (i = 1,2,...,n), le but d'une méthode d' évaluation ”filter” est de calculer un score pour évaluer la pertinence de chacune des x^i.

Les mesures présentées ci-dessous sont des scores ou critères d'évaluation utilisés

dans la littérature.

⇒ **Le critère de corrélation :** utilisé dans le cas d'une classification binaire $y_k \in \{1, 1\}$. Il est calculé comme suit [36, 50] :

$$C(i) = \frac{\sum_{k=1}^{m}(x_{ki} - \mu_i)(y_k - \mu_y)}{\sqrt{\sum_{k=1}^{m}(x_{ki} - \mu_i)^2 \sum_{k=1}^{m}(y_k - \mu_y)^2}} \qquad [36] \qquad (II.1)$$

où μ_i et μ_y sont respectivement les valeurs moyennes du $i^{ème}$ attribut et des étiquettes de l'ensemble d'apprentissage. Il calcule le cosinus de l'angle entre chaque attribut et le vecteur des étiquettes. Une grande valeur absolue pour ce critère indique la forte corrélation linéaire de l'attribut avec le vecteur des étiquettes (Y).

⇒ **Le critère de Fisher :** permet de mesurer le degré de séparabilité des classes à l'aide d'un attribut donné. Il est défini par [36] :

$$F(i) = \frac{\sum_{c=1}^{C} \eta_c (\mu_c^i - \mu^i)^2}{\sum_{c=1}^{C} \eta_c (\sigma_c^i)^2} \qquad [36] \qquad (II.2)$$

où η_c , μ_c^i et σ_c^i désignent respectivement l'effectif, la moyenne et l'écart type de la $i^{ème}$ caractéristique au sein de la classe c. μ^i est la moyenne globale de la $i^{ème}$ caractéristique. Cette mesure est liée à la variance interclasse de la caractéristique.

⇒ **L'information mutuelle** : évalue la dépendance entre les distributions de deux populations. Soient X et Y deux variables aléatoires dont les instances sont respectivement les valeurs de la $i^{ème}$caractéristique et les étiquettes des classes. L'information mutuelle MI (i) peut être considérée comme la divergence de Kullback-Leibler (KL) entre la probabilité $P(x^i, y)$ et le produit des probabilités $P(x^i, P(y))$. L'information mutuelle est estimée empiriquement par [15] :

$$MI(i) = \sum_{x_i} \sum_{y} P(X = x_i, Y = y) \log \frac{P(X = x_i, Y = y)}{P(X = x_i)P(Y = y)} \quad [36] \quad (II.3)$$

où les probabilités P(x_i), P(y) et P(x_i, y) sont estimées par les fréquences des différentes valeurs possibles.

1.2.5.2 Les méthodes de type enveloppantes (wrapper method)

Le principal inconvénient des approches ”filter” est le fait qu'elles ignorent l'influence des attributs sélectionnées sur la performance du classificateur à utiliser par la suite. A l'inverse des approches « filtres », les méthodes « wrapper » nécessitent le choix préalable d'une méthode de classification pour être appliquées [35, 48].

La qualité d'un sous-ensemble de variables *D* est quantifiée par le taux d'erreur obtenu en combinant *D* avec l'algorithme de classification choisi. On dit alors que la méthode de sélection est dédiée à une méthode de classification donnée. L'évaluation se fait à l'aide d'un classifieur qui estime la pertinence d'un sous-ensemble donné de caractéristiques. La figure suivante explique ce principe.

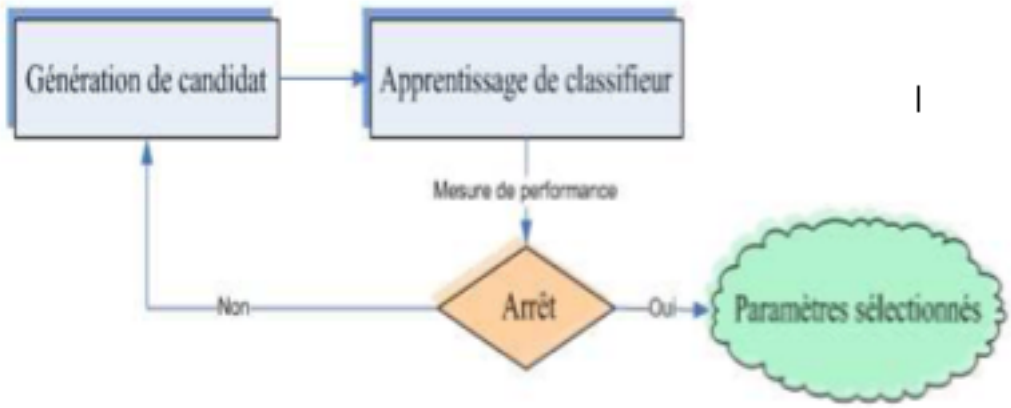

Figure I.3 - Principe de l'approche enveloppante [36].

Le problème de la complexité de cette technique rend impossible l'utilisation d'une stratégie de recherche exhaustive (problème NP-complet). Par conséquent, des

méthodes de recherche heuristiques ou aléatoires peuvent être utilisées. La recherche devient néanmoins, de plus en plus irréalisable avec l'augmentation de la taille de l'ensemble initial de caractéristiques.

Les stratégies "wrapper" sont généralement considérées comme étant meilleures en termes de performance de classifieur que les méthodes « filtres ». Elles permettent de sélectionner des sous-ensembles d'attributs de petite taille performants pour le classifieur utilisé mais présentent les inconvénients suivants :

- ✓ L'ensemble des paramètres est spécifique au classifieur utilisé. Ainsi le sous-ensemble sélectionné dépend toujours du classificateur utilisé. Or chaque classificateur a ses spécificités et ses hypothèses. Donc nous ne pouvons pas garantir que l'ensemble trouvé donne de bons résultats avec d'autres algorithmes de classification.

- ✓ Le calcul du taux de classification est très coûteux du point de vue temps de calcul parce que les sous-ensembles sont évalués par un classifieur. Ainsi cette approche demande un temps de calcul énorme pour la sélection des attributs comparée à l'approche de filtre.

1.2.5.3 Les méthodes intégrées (embedded methods)

Ces méthodes sont proches des méthodes enveloppantes, car elles combinent le processus d'exploration avec un algorithme d'apprentissage. La différence avec les méthodes enveloppantes est que le classifieur sert non seulement à évaluer un sous-ensemble candidat mais aussi à guider le mécanisme de sélection. L'avantage de ces méthodes est que le processus de recherche est guidé par des informations intéressantes fournies par le classifieur, ce qui rend ces méthodes plus efficaces que les méthodes enveloppantes [36, 48].

Les méthodes intégrées contrairement aux méthodes enveloppantes qui divisent la base d'apprentissage en deux parties : une base d'apprentissage et une base de validation pour valider le sous-ensemble de caractéristiques sélectionné, peuvent se servir de tous les exemples d'apprentissage pour établir le système. Cela peut donc

aboutir à une amélioration des résultats. Ces méthodes présentent également l'avantage d'être plus rapides par rapport aux approches enveloppantes parce qu'elles évitent que le classifieur recommence de zéro pour chaque sous-ensemble de caractéristiques.

Parmi les derniers travaux qui utilisent cette approche, on peut citer les travaux dans lesquels le mécanisme d'apprentissage fournit des poids aux attributs pour faire la sélection. On peut citer notamment RFE-SVM.

1.3 Revue de différentes méthodes de sélection d'attributs classiques

Il existe dans la littérature un nombre considérable de méthodes de sélection d'attributs. Nous allons présenter ici différentes méthodes basées sur les stratégies de recherche et évaluation vues plus haut.

1.3.1 SFS et SBS

La méthode SFS est une méthode enveloppante populaire qui utilise une approche heuristique de recherche, en partant d'un ensemble vide d'attributs. Le meilleur attribut est ajouté au sous ensemble d'attributs sélectionné et supprimé de l'ensemble d'origine à chaque itération. Le processus de sélection continue jusqu'à un critère d'arrêt. Par la suite la méthode SBS (Sequential Backward Selection) a été créée. Cette méthode à l'inverse de SFS commence par l'ensemble de toutes les caractéristiques et supprime à chaque itération, la caractéristique la moins pertinente [36, 49].

Des auteurs ont prouvé que la méthode SBS est plus performante parce qu'elle tient compte de l'interaction d'un attribut avec un ensemble d'attributs plus large alors que SFS ne prend en considération que l'interaction de cet attribut avec le sous-ensemble déjà sélectionné [36, 49]. Toutefois SBS prend trop de temps pour évaluer les sous-ensembles de grande taille.

GSFS et GSBS sont des généralisations des méthodes SBS et SFS qui ont été proposées [36, 50]. Elles permettent d'inclure (ou exclure) un sous ensemble

d'attributs à chaque itération au lieu d'en inclure ou d'en exclure un. Ces méthodes sont plus performantes que les méthodes initiales mais présentent toujours les mêmes problèmes que les méthodes de base.

SFFS (Sequential Floating Forward Selection) et SFBS (Sequential Floating Backward Selection) ont été proposées en 1994 [36, 50] afin de limiter les inconvénients des méthodes décrites ci-dessus. Elles consistent à utiliser x fois l'algorithme SFS pour ajouter x attributs, puis à utiliser y fois l'algorithme SBS pour en supprimer y. Le processus est répété jusqu'à l'obtention du critère d'arrêt. La taille du sous-ensemble à chaque étape dépend alors des valeurs de x et y. Théoriquement, les valeurs optimales de ces paramètres ne peuvent pas être déterminées. C'est pour cette raison que les auteurs suggèrent de les laisser flottantes au cours du processus de sélection dans le but de se rapprocher le plus possible de la solution optimale.

Algorithme SFS

Entrées :

$F = \{f_1, f_2, .., f_N\}$

M : taille de l'ensemble final

Sorties : $S = \{f_{s1}, f_{s2}, .., f_{sM}\}$

$S = \emptyset$

Pour i = 1 à M **Faire**

  Pour j = 1 à |F | **Faire**

    Evaluer $f_j \cup S$

  Fin Pour

  f_{max} = meilleure f_j

  $S = S \cup f_{max}$, $F = F \setminus f_{max}$

Fin Pour

Retourner S

Algorithme SBS

Entrées :

F = {f_1, f_2, .., f_N }

M : taille de l'ensemble final

Sorties : S = {f_{s1}, f_{s2}, .., f_{sM} }

S = F

Pour i = 1 à N-M **Faire**

Pour j = 1 à |S| **Faire**

Evaluer S \ f_j

Fin Pour

f_{min} = la plus mauvaise f_j

S= S \ f_{min}

Fin Pour

Retourner S

1.3.2 Branch and Bound

La méthode "Branch and Bound" (BB) [36, 50] utilise une fonction qui permet de mettre une borne sur certaines solutions pour soit les exclure, soit les maintenir comme des solutions potentielles. Cette méthode garantit de trouver un sous-ensemble optimal d'attributs si une fonction d'évaluation monotone est utilisée. Cette contrainte limite bien sûr le choix de la mesure de distance. Généralement, celles qui sont utilisées sont la distance de Mahalanobis [36, 50], la distance de Bhattacharya [36, 50], le critère de Fisher [36, 40], la fonction discriminante et la divergence [36, 50].

Cette méthode construit un arbre de recherche dans lequel la racine représente l'ensemble des attributs et les autres nœuds représentent des sous-ensembles d'attributs. En parcourant l'arbre de la racine aux feuilles, l'algorithme enlève successivement le plus mauvais attribut du sous ensemble courant (nœud courant) qui ne satisfait pas le critère de sélection. Si la valeur attribuée à un nœud est plus petite qu'un seuil (bound), les sous-arbres de ce nœud sont supprimés.

Cette approche, couvrant une grande partie de l'espace des solutions, nécessite un nombre élevé d'opérations. Elle apporte tout de même une réduction par rapport à une approche exhaustive. Mais le temps de calcul croît vite avec l'augmentation du

nombre d'attributs et devient impraticable à partir d'un certain nombre (30 attributs). Par la suite une amélioration de cette méthode a été proposée en utilisant d'autres techniques de recherche dans l'arbre afin d'accélérer le processus de sélection [36, 50].

1.3.3 FOCUS

FOCUS [36,50] a été proposé en 1991 et repose sur une recherche exhaustive sur l'ensemble initial d'attributs afin de trouver le sous ensemble le plus performant de taille minimale. Cette méthode commence par générer et évaluer tous les sous-ensembles de taille N (un au départ), puis tous les couples d'attributs, les triplets et ainsi de suite jusqu'à ce que le critère d'arrêt soit satisfait. Mais cette méthode ne fonctionne pas correctement avec les données bruitées. De plus le temps de calcul devient trop grand avec l'augmentation de la taille de l'ensemble des attributs et du nombre d'instances de la base. Il existe aussi une variante de cet algorithme appelé FOCUS2 qui est beaucoup plus rapide que FOCUS mais reste toujours sensible au bruit.

Algorithme FOCUS

Entrées: Une base d'apprentissage B = $\{X_1, X_2, .., X_M\}$ où $X_i = \{x_{i1}, x_{i2}, .., x_{iN}\}$,
K : Taille maximale de l'ensemble final et un seuil σ

Sorties: F : ensemble final des attributs

```
F = ∅
Pour i = 1 à T Faire
   Pour chaque sous-ensemble (F1) de taille (i) Faire
         Cons = Inconsistance (A, F1)
         Si Cons < σ alors
            F = F1
            Retourner F
         Fin Si
    Fin Pour
Fin Pour
```

1.3.4 RELIEF

Le principe de la méthode relief [36, 45, 47] est de procéder à une pondération des différents attributs. L'utilisateur doit au préalable fixer le nombre d'échantillons que l'algorithme choisira aléatoirement dans le corpus d'apprentissage. Ensuite pour chacun de ces échantillons l'algorithme recherche au sein de ce sous-ensemble l'échantillon de la même classe le plus proche et celui d'une classe différente la plus proche aussi. On calcule alors une mesure globale de la pertinence des attributs en accumulant la différence des distances entre ces exemples choisis aléatoirement et leurs plus proches voisins de la même classe et de l'autre classe. Les poids des attributs sont mis à jour en fonction de ces valeurs. A la fin du processus, les attributs sélectionnés sont ceux qui ont une pondération supérieure à un seuil. Cette méthode présente comme avantage sa simplicité, sa facilité de mise en œuvre ainsi que sa précision même sur des données bruitées. Par contre, sa technique aléatoire ne peut pas garantir la cohérence des résultats lorsqu'on applique plusieurs fois la méthode sur les mêmes données. Par ailleurs, cette méthode ne détecte pas la présence d'attributs redondants. Une restriction importante à souligner est que cet algorithme ne fonctionne qu'avec 2 classes. Une variante de cet algorithme appelée Relief-F qui propose plusieurs classes a été proposée par la suite. D'autres variantes de cet algorithme, pour améliorer sa performance, sa vitesse ou les deux, ont été également proposées.

Algorithme Relief

Entrées : Une base d'apprentissage B = $\{X_1, X_2, .., X_M\}$ où chaque exemple $X_i = \{x_{i1}, x_{i2}, .., x_{iN}\}$

Nombre d'itérations L

Sorties: W [N] : vecteur de poids des attributs (f_i), $-1 \leq W[i] \leq$

1 $\forall i$, W [i] = 0;

Pour t = 1 à L **Faire**

Choisir aléatoirement un exemple X_k

Chercher deux plus proches voisins (un dans sa classe (X_a) et un deuxième dans l'autre classe (X_b))

Pour i=1 à N **Faire**

$$W[i] = W[i] + \frac{|x_{ki} - x_{bi}|}{M \times T} - \frac{|x_{ki} - x_{ai}|}{M \times T}$$

Fin pour

Fin Pour

Retourner W

1.3.5 LVW et LVF

La méthode LVW (Las Vegas Wrapper) génère aléatoirement à chaque itération, un sous-ensemble d'attributs et l'évalue avec un classificateur [36, 45].

Après avoir évalué, si sa performance est meilleure que la meilleure performance trouvée auparavant (au départ, l'ensemble de base est supposé comme le meilleur sous-ensemble), ce sous-ensemble devient le meilleur sous-ensemble courant. Ce processus est répété jusqu'à ce que T essais consécutifs soient infructueux pour l'amélioration. Cette méthode présente l'inconvénient de ne pas garantir l'optimalité de la solution finale ainsi qu'un temps de calcul très élevé.

LVF (Las Vegas Filter) est une méthode « filtre » similaire à LVW et évalue des sous-ensembles par le calcul d'une mesure appelée "taux d'incohérence" ou "taux d'inconsistance". L'inconsistance pour un sous-ensemble d'attributs est définie par le rapport entre le nombre d'exemples inconsistants de la base de données et le nombre total d'exemples. Un exemple est dit inconsistant s'il existe un autre exemple qui a la même représentation dans l'espace des attributs du sous-ensemble d'attributs étudié (appelé exemple équivalent), mais qui appartient à une autre classe. Cette méthode est très sensible au bruit et est très coûteuse en termes de temps de calcul.

Algorithme LVW

Entrées : Une base d'apprentissage B

Une base d'attributs F

Nombre d'itérations L

Sorties : F : Ensemble sélectionné

Err = Classificateur (B, F)

k = 0, N = |F|

Répéter

F1 = Genérer_Al(),N1 = |F1|

Err1 = Classificateur (B, F1)

Si (Err1 < Err) ou (Err1 = Err et N1 < N) **alors**

k = 0, N = N1, F = F1, Err = Err1

Fin Si

k = k + 1

Jusqu'à k=L

Retourner F

Algorithme LVF

Entrées : Une base d'apprentissage B

Une base d'attributs F

Nombre d'itérations L et un seuil σ

Sorties : F : Ensemble sélectionné

N = |F|

Pour i=1 à L

F1 = Genérer_Al ()

N1 = |F1|

Err1 = Classificateur (B, F1)

Si Inconsistance (A, S1) < σ et (N1 < N) **alors**

N = N1

S = S1

Fin Si

Fin pour

Retourner S

1.3.6 Max-relevance, Min-Redundancy (mRMR)

mRMR (maximum Relevancy Minimum Redundancy) est une méthode « filtre » qui consiste à sélectionner les attributs de façon à ce qu'ils soient mutuellement au maximum dissemblables et au maximum pertinentes avec l'étiquette de classe l [36, 45]. Cette méthode est basée sur des mesures statistiques classiques comme l'information mutuelle, la corrélation, etc.

Soit F l'ensemble des attributs. Soit f_i et f_j deux variables dans F. $MI(f_i, f_j)$ représente la mesure de l'information mutuelle entre les variables f_j et f_i. $MI(l, f_i)$ est la mesure de l'information mutuelle entre l'étiquette de classe l et f_i.

La redondance parmi les attributs de F, qui est déterminée par l'information mutuelle entre eux est donnée par [12, 36] :

$$Q_I(F) = \frac{1}{|F|^2} \sum_{f_i, f_j \in F} MI(f_i, f_j) \qquad [36] \qquad (II.4)$$

La pertinence des attributs de F avec l'étiquette de classe l est calculée comme suit [12]:

$$R_I(F) = \frac{1}{|F|} \sum_{f_i \in F} MI(l, f_i) \qquad [36] \qquad (II.5)$$

L'ensemble d'attributs au maximum pertinents et au minimum redondants parmi tous les ensembles F ' dans F est obtenu en optimisant (II.4) et (II.5) comme suit [36, 50]:

$$F^* = \text{argmax}_{F' \subseteq F} [R_I(F) - Q_I(F)] \qquad [36] \qquad (II.6)$$

Parmi la vaste gamme de méthodes FS disponibles, la méthode de filtrage mRMR a été l'une des plus utilisées ces dernières années. Plusieurs auteurs rapportent sa grande popularité (grâce à sa précision), en dépit d'être une méthode de calcul coûteuse. Ils montrent également son utilité dans de nombreux problèmes. Cette méthode évolue de manière quadratique avec le nombre de caractéristiques.

1.3.7 RFE-SVM (Recursive Feature Elimination- Support Vector Machine)

RFE-SVM (Recursive Feature Elimination-Support Vector Machine) est une technique efficace de sélection de caractéristiques qui permet de réduire le nombre d'attributs utilisés pour l'analyse ou le développement de modèles de prédiction. Dans l'analyse des données des puces à ADN, cette technique est couramment utilisée en particulier dans la découverte de maladies génétiques afin d'éliminer les gènes redondants et donne des sous-ensembles de gènes meilleurs et plus compacts. Les caractéristiques sont éliminées selon un critère lié à leur poids SVM, et le SVM est recyclé à chaque étape [35].

La pertinence des attributs de F avec l'étiquette de classe l est calculée en se basant sur l'hyperplan qui a la plus grande distance avec les points de données voisins des deux classes. L'espoir est que, plus la marge ou la distance entre ces hyperplans parallèles est grande, meilleure sera l'erreur de généralisation du classifieur [35].

Algorithme RFE-SVM

RFE-SVM suit quatre étapes.

1. Lancer le classifieur SVM sur l'ensemble d'apprentissage;

2. Ordonner les attributs en utilisant les poids résultant du classifieur ;

3. Éliminer les attributs avec les plus petit poids ;

4. Répétez le processus avec l'ensemble d'apprentissage restreint aux autres attributs.

Le tableau suivant récapitule les méthodes de sélection d'attributs classiques présentés ici.

Méthode	Type	Stratégie de recherche	Non elimination de la redondance	Non prise en compte ses interactions	Complexité	Dépendance à la fonction d'´evaluation	Sensibilité aux bruits
RFE-SVM	Wrapper	Heuristique					
mRMR	Filter	Heuristique					
LVF	Filter	Aléatoire	X		X		X
LVW	Wrapper	Aléatoire			X	X	
Relief	Filter	Aléatoire	X	X			
Focus	Filter	Exhaustive		X			X
B and B	Filter ou Wrapper	Heuristique			X	X	
SBS	Filter	Heuristique	X		X		
SFS	Filter	Heuristique	X	X			

Tableau II.2 –Résumé des méthodes de sélection d'attributs présentées [36]

2. Sélection d'instances

Généralement, il y a des instances « bruyantes » ou superflues dans les ensembles de données et il est donc nécessaire de les supprimer de l'ensemble pour une étude donnée. Les méthodes de sélection d'instances permettent de réduire la taille des données en supprimant les instances non pertinentes afin d'améliorer les performances d'un algorithme d'apprentissage basé sur les instances. Dans les classifieurs basés sur des instances, les techniques de sélection d'instances doivent être capables de réduire le temps d'apprentissage d'un classifieur et obtenir des taux de classification identiques ou même meilleurs que ceux obtenus avec l'ensemble complet de données [51, 34]. Les méthodes de sélection d'instances classiques peuvent suivre le même processus que celui des méthodes de sélection d'attributs comme expliqué dans la section **1.2**.

Ainsi, tout comme avec les méthodes de sélection d'attributs enveloppantes, les techniques de sélection d'instance enveloppantes sélectionnent les instances à l'aide d'un modèle de classification et dépendent de la précision obtenue par le classificateur. Des tests sont effectués séparément dans les blocs d'instances de l'ensemble de données en entraînant un modèle. Enfin, la précision de chaque modèle est évaluée et le sous-ensemble avec la plus grande précision est sélectionné [52].

Les techniques de sélection d'instances « filtres » ne sont généralement pas dépendantes d'un classifieur. Ces méthodes évaluent les instances selon des heuristiques basées sur les caractéristiques globales des données afin d'identifier les instances qui peuvent être supprimées des données d'apprentissage en toute sécurité [53]. Les techniques de sélection d'instances « filtres » sont généralement plus rapides que les techniques enveloppantes. Les méthodes « filtres » sont plus générales que les méthodes enveloppantes et ne sont pas étroitement associées à un algorithme d'apprentissage [54]. C'est la raison pour laquelle nous proposons dans ce mémoire une méthode de sélection d'instances « filtre ».

Définition II.2 : La sélection d'instances peut être formalisée comme suit : Soit D un ensemble d'apprentissage avec un grand nombre d'instances I, Soit f (I', D) la fonction objective qui évalue le sous-ensemble I' de I en utilisant les données de D.

La sélection d'instances consiste à trouver un sous-ensemble S d'instances de D qui maximise la fonction d'évaluation f. Ce sous-ensemble est alors considéré comme le meilleur sous-ensemble. Un sous-ensemble I_1 est meilleur que I_2 si f (I_1, D)> f (I_2, D).

2.1 Revue de différentes méthodes de sélection d'instances

En raison de l'augmentation de la taille des ensembles de données, des techniques de sélection d'instances ont été appliquées pour réduire les données à un volume gérable, conduisant à une réduction des ressources de calcul qui sont nécessaires pour effectuer le processus d'apprentissage. Au cours des dernières années, plusieurs approches de sélection d'instances ont été proposées [55]. Nous allons présenter ici les principales méthodes de sélection d'instances classiques qui existent dans la littérature.

Toutes ces méthodes de sélection d'instances ont une particularité en commun : elles ne sont adaptées qu'au classifieur k-NN (k plus proches voisins). Ainsi nous commencerons par présenter k-NN avant de passer à la présentation des méthodes de sélection d'instances classiques de l'état de l'art.

⇒ **k-NN**. La méthode des k plus proches voisins (k-NN) est une méthode d'apprentissage supervisé. Il est abrégé k-NN ou KNN, de l'anglais k-Nearest Neighbor [36].

Pour utiliser k-NN, on doit disposer d'une base de données d'apprentissage constituée de N couples « entrée-sortie ». La méthode des k-NN consiste alors à prendre en compte les k échantillons d'apprentissage dont l'entrée est la plus proche d'une nouvelle entrée x, selon une distance à définir, afin de déterminer la sortie associée à une nouvelle entrée x.

Par exemple, dans un problème de classification, on retiendra la classe la plus représentée parmi les k sorties associées aux k entrées les plus proches de la nouvelle entrée x.

Nous allons maintenant passer à la présentation des méthodes de sélection d'instances classiques de l'état de l'art. Il s'agit de :

⇒ **CDIS (ou LDIS)**

La plupart des méthodes de sélection d'instances ne sont pas utilisables dans les grands ensembles de données parce qu'elles présentent une grande complexité temporelle. C'est pour cette raison que l'algorithme CDIS a été créé [53, 55]. Il peut être considéré comme une amélioration d'une approche basée sur la densité récemment proposée pour la sélection d'instances. Cette technique utilise une nouvelle fonction de densité et évalue les instances de chaque classe séparément en ne conservant que les instances les plus denses dans un voisinage donné (arbitraire). Cela garantit une complexité temporelle raisonnablement faible. Cette approche a été évaluée sur vingt ensembles de données bien connus et sa performance a été comparée à celle de six algorithmes de pointe, en considérant trois mesures : la précision, la réduction et l'efficacité. Pour évaluer la précision obtenue en utilisant les jeux de données produits par les algorithmes, le classifieur utilisé est KNN. Les résultats montrent que cette approche atteint une performance (en termes d'équilibre de précision et de réduction) meilleure ou comparable aux performances des autres algorithmes considérés dans l'évaluation.

⇒ **CNN**

Cet algorithme utilise la méthode Condensed Nearest Neighbours ou la méthode Density Preserving Sampling et est capable de réduire les besoins en mémoire de l'ensemble d'apprentissage à 5% de sa taille d'origine sans affecter les performances [57]. Le stockage supplémentaire requis pour l'étape de la classification avec KNN devient alors beaucoup plus petit.

⇒ **LSBo**

L'ensemble local est la plus grande hypersphère centrée sur une instance telle qu'elle ne contienne aucune instance d'une autre classe. En raison de sa nature géométrique, cette structure peut être très utile pour la classification basée sur la distance, telle que la classification basée sur la règle du plus proche voisin. C'est pour cela que la méthode **LSBo** est axée sur la sélection d'instances pour la classification avec KNN qui, en bref, vise à réduire le nombre d'instances dans l'ensemble d'apprentissage sans affecter la précision de la classification. Ainsi les auteurs de **LSBo** proposent trois méthodes de sélection d'instances basées sur des ensembles locaux, qui suivent des stratégies différentes et complémentaires [54]. Dans une étude

expérimentale impliquant vingt-six bases de données connues, ils sont comparés à onze des méthodes les plus performantes de pointe dans des environnements standard et « bruyants ». Les résultats obtenus par les propositions révèlent qu'elles figurent parmi les méthodes les plus efficaces dans ce domaine.

⇒ **SV-kNNC**

SV-kNNC est un nouvel algorithme destiné au classifieur k-Nearest Neighbor (KNN) [56]. Il est composé de trois étapes. Tout d'abord, des classifieurs SVMs sont utilisés pour sélectionner certaines données importantes de l'ensemble d'apprentissage. Ensuite, la méthode k-mean est utilisée pour attribuer un poids à chacune de ces instances. Enfin, des exemples non vus sont classés par KNN. SV-kNNC fournit de meilleurs résultats que la méthode CNN, à la fois en termes de précision prédictive que de temps de classification.

Le tableau suivant récapitule les méthodes sélection d'instances classiques présentées dans l'état de l'art.

Algorithme	**Insuffisances**	**Reference**
CDIS	N'est adapté qu'au classifieur k-NN	[53]
LSBo	N'est adapté qu'au classifieur k-NN	[54]
LDIS	N'est adapté qu'au classifieur k-NN	[55]
SV-kNNC	N'est adapté qu'au classifieur k-NN	[56]
CNN	N'est adapté qu'au classifieur k-NN	[57]

Table II.3 - Résumé de l'état de l'art des méthodes de sélection d'instances non parallèles

3. Conclusion

Dans ce chapitre nous avons défini la sélection d'attributs et la sélection d'instances et montré leurs principes de fonctionnement. Nous avons également

présenté des méthodes classiques qui existent dans la littérature pour chacune de ces techniques.

Mais le problème de ces méthodes est qu'avec la taille actuelle des données, elles posent un problème de passage à l'échelle ; elles ne sont donc pas adaptées aux grandes masses de données. Pour faire face à cette problématique, la parallélisation dans des environnements comme Hadoop ou Spark pourrait être une solution.

Aussi dans le chapitre qui suit, nous allons présenter les méthodes de sélection d'attributs et de sélection d'instances **parallèles** qui ont été proposées dans la littérature.

Chapitre III

Passage à l'échelle des algorithmes de réduction de données

Avec la croissance explosive des données, un processus centralisé de réduction de données devient incapable de traiter de gros volumes de données. La distribution du calcul sur plusieurs machines a résolu le problème. C'est-à-dire que les données sont situées sur des machines différentes, et le processus de calcul est réparti sur ces données partagées et est exécuté en parallèle [1].

Dans ce chapitre nous présenterons plusieurs méthodes de réduction de données parallèles qui ont été présentées dans la littérature.

1. Passage à l'échelle des algorithmes de la sélection d'attributs

Dans le but d'adapter la sélection d'attributs aux problèmes de BIG DATA, des auteurs ont fait des propositions que nous présentons ici.

⇒ **Generic FS framework :**

Ce framework a été proposé dans [1] et représente une parallélisation dans Apache Spark, d'un large groupe de méthodes bien connues y compris mRMR et qui sont basées sur la théorie de l'information.

Les résultats expérimentaux sur un large éventail de jeux de données du monde réel indiquent des performances compétitives (en termes de généralisation et d'efficience) dans le cas de jeux de données de très grandes tailles tant en termes de nombre d'attributs que d'instances.

Les auteurs ont également analysé l'utilité de leur solution lorsqu'elle est appliquée à une classification à grande échelle. Les résultats de précision indiquent une

amélioration importante pour la plupart des jeux de données lorsque la sélection d'attributs a été utilisée.

Enfin, l'évolutivité de la méthode a aussi été étudiée, en faisant varier le nombre de cœurs (de 10 à 100), les résultats révèlent un comportement logarithmique à mesure que le nombre de cœurs augmente.

Dans la version originale de mRMR, le goulot d'étranglement principal est lié au calcul de l'information mutuelle entre deux éléments : soit une caractéristique donnée avec la classe, soit une paire d'attributs en entrée. Pour faire face à ce problème, les auteurs proposent ici une approche gloutonne pour mRMR qui utilise le nombre d'attributs à sélectionner afin de limiter le nombre de comparaisons entre attributs. Cet algorithme est décrit comme suit :

1. Les **workers** calculent d'abord les valeurs de pertinence des attributs en entrée.
2. Les **workers** envoient les résultats au driver.
3. Le **driver** sélectionne le meilleur attribut selon la pertinence et l'ajoute dans l'ensemble d'attributs à retourner.
4. Le **driver** demande aux **workers** de supprimer l'attribut sélectionné de leur partition.
5. Les **workers** calculent la redondance entre l'attribut sélectionnée par le **driver** et le reste des attributs dans leur partition, puis mettent à jour le critère mRMR des attributs.
6. Les **workers** envoient à nouveau les résultats au **driver**.
7. Le **driver** sélectionne l'attribut qui a le plus grand score mRMR et l'ajoute dans l'ensemble d'attribut à retourner.

Le processus est répété de la **5ème étape** à la **7ème** jusqu'à l'obtention du nombre d'attributs désiré.

Insuffisance : l'algorithme est couteux parce que le nombre d'aller retours entre les workers et le driver augmente en fonction du nombre d'attributs à sélectionner.

Ce qui peut entrainer des couts de communication et de synchronisation élevés entre les workers et le driver.

⇒ **Manchester AnalyticS Toolkit (MAST)** [71] **:**

Il s'agit d'une boîte à outils qui fournit une implémentation efficace, parallèle et évolutive de techniques de sélection d'attributs basées sur la théorie de l'information.

Dans la méthode utilisée par les auteurs, pour calculer les informations mutuelles, la probabilité de chaque combinaison de caractéristiques et de valeurs d'étiquettes est estimée. C'est-à-dire, pour chaque paire de caractéristiques X et Y avec l'étiquette L, il faut estimer la probabilité de la combinaison (xi, yj, lk). Une méthode efficace pour estimer cette probabilité consiste à adopter la méthode de l'histogramme basée sur le comptage de la fréquence des occurrences dans les données. Parce que les estimations de probabilité sont souvent nécessaires, MAST fournit des structures de données appelées objets JointRandomVariable (jrv) qui stockent les comptages de fréquence et fournissent des mesures théoriques de l'information.

Les auteurs utilisent plusieurs structures pour stocker les jrv, dont une appelée tableau à jetons qui utilise moins de mémoire et est d'un ordre de grandeur plus rapide que les autres structures proposées. Pour ce faire, les données sont prétraitées en trois étapes. D'abord les données sont lues en blocs. Ces blocs sont ensuite transposés de sorte que les valeurs consécutives de la même caractéristique se trouvent dans des blocs de mémoire contigus. Enfin, les valeurs de caractéristiques sont converties en "jetons", les jetons constituent des entiers consécutifs. Le prétraitement a comme objectif de réduire la mémoire utilisée mais entraîne une augmentation du temps d'exécution dans certains cas.

Insuffisances :

-Les auteurs ne donnent pas de détail sur leur méthodologie d'optimisation de la mémoire.

-Ils n'ont pas donné de code source.

-Les Jeux de données utilisés pour les expériences n'ont pas été précisés.

-Il faut souligner aussi que les auteurs de MAST n'ont pas analysé la performance de leurs méthodes en termes de précision pour une éventuelle phase d'apprentissage qui suivrait la sélection d'attributs. Ainsi leur proposition concerne l'amélioration du temps d'exécution et de l'utilisation de la mémoire.

⇒ **Fast-mRMR** [72] **:**

Cette technique est une extension de mRMR proposée pour surmonter le calcul coûteux de mRMR. Les auteurs fournissent un paquet avec trois implémentations de cet algorithme dans plusieurs plates-formes, à savoir, CPU pour l'exécution

séquentielle, GPU (unités de traitement graphique) pour le calcul parallèle, et Apache Spark pour le calcul distribué. **Fast-mRMR** surpasse la version originale de mRMR.

Cet algorithme est celui que nous avons décrit précédemment dans Generic FS framework où il a été intégré [1]. Ainsi les insuffisances sont ceux citées précédemment pour **Generic FS framework**.

Contributions :

De nombreux efforts de recherche [73, 72, 1, 72] ont été introduits pour concevoir des algorithmes parallèles capables de travailler dans un environnement de mémoire partagée. La plupart d'entre eux sont très performants et passent à l'échelle.

Cependant ces méthodes utilisent une approche gloutonne afin de limiter le nombre de comparaisons entre attributs. Ainsi, au départ on a un sous-ensemble d'attributs à retourner qui est vide. Puis à chaque itération l'attribut considéré comme le meilleur de l'ensemble par la méthode mRMR est ajouté à l'ensemble des attributs à retourner. Le processus s'arrête une fois que le nombre d'attributs à retourner est obtenu. Donc l'algorithme itère autant de fois qu'il y a d'attributs à retourner.

De plus ces méthodes utilisent plusieurs jobs map-reduce pour effectuer la sélection d'attributs. Or généralement, dans un environnement massivement distribué, la performance globale d'un processus est améliorée quand on peut minimiser le nombre de "jobs" (les "aller/retours" entre les machines distribuées). Cela impacte le temps d'exécution, mais aussi le transfert de données. Dans le cas de la réduction de données, trouver les meilleurs attributs en un seul job simplifié serait donc préférable.

A noter aussi que parmi ces méthodes rares sont celles qui évaluent à la fois la performance du classifieur utilisé après la sélection des attributs et le temps d'exécution de la méthode. C'est le cas par exemple avec Manchester Analytics Toolkit.

Voilà pourquoi nous introduisons de nouvelles méthodes qui tiennent compte à la fois du temps d'exécution et de la précision du classifieur. Notre objectif est de réduire le temps d'exécution en minimisant les communications et synchronisations entre workers et driver tout en améliorant les performances des classifieurs. Pour cela notre stratégie consiste à calculer le score de chaque attribut au niveau des workers et à envoyer les résultats au driver qui sélectionne en une fois les meilleurs attributs. Nos méthodes « filtres » effectuent ainsi la sélection d'attributs en un seul job et procèdent à une seule itération pour sélectionner les attributs à retourner. Les

expériences ont montré que nos méthodes sont très efficientes et entrainent une diminution considérable du temps d'exécution des algorithmes de sélection d'attributs. De plus nos algorithmes montrent de meilleures performances en terme de précision en comparaison avec la plupart des algorithmes parallèles cités plus haut.

Dans l'une des méthodes que nous avons proposées pour paralléliser l'algorithme mRMR, nous améliorons la métrique utilisée par cette dernière dans le but d'accroitre la précision de la classification. Notre métrique basée sur les SVM (machines à vecteurs de support) est inspirée de [76]. Le choix de SVM est dû à ses performances qui ont été décrites par de nombreux auteurs dans la littérature [77]. Notre méthode présente de très bonnes précisions surtout en comparaison avec les versions parallèles proposées dans la littérature.

Dans l'algorithme PNFS_Spark qui fait partie des techniques que nous proposons aussi, nous utilisons une toute nouvelle métrique basée sur l'utilisation de la valeur médiane de la pertinence des attributs avec l'étiquette de classe. En fait, la médiane donne une idée satisfaisante de la tendance générale d'une série statistique (l'ensemble des attributs dans notre cas).

Dans cette méthode, la relevance de chaque attribut est d'abord calculée, puis la médiane de ces valeurs que nous appelons R_{fe} est déterminée. Par la suite, pour chaque attribut, ses voisins sont déterminés. Un attribut f_i est considéré comme le voisin d'un autre attribut f_j si l'information mutuelle M_{ij} entre les 2 est supérieure à la valeur médiane R_{fe}. En effet nous considérons que cela indique une forte corrélation entre les deux attributs f_i et f_j. Si nV est le nombre de voisins d'un attribut f_i et sa relevance R_i alors le score final r_i de l'attribut f_i sera : $nV*R_i$. Nous considérons que l'attribut qui possède la relevance R_i qui détermine son degré de pertinence, est pertinent autant de fois qu'il a de voisins. Donc plus il a de voisins, plus son importance augmente. Les attributs qui ont obtenu les meilleurs scores sont renvoyés par notre algorithme.

2. Passage à l'échelle des algorithmes de la sélection d'instances

La sélection d'instances est une tâche de prétraitement populaire dans la découverte des connaissances et l'exploration de données. Son but est de réduire la taille des ensembles de données tout en conservant leurs capacités prédictives. Le problème émergent habituel à ce stade est que ces méthodes souffrent souvent d'une complexité de calcul élevée, ce qui devient très gênant pour le traitement d'énormes ensembles de données [78].

Pour faire face au BIG DATA, des propositions ont été présentées récemment dans la littérature (voir table **III.1**). Tous ces algorithmes à grande échelle développent la même idée : obtenir un volume gérable à partir des BIG DATA en réduisant la mémoire nécessaire pour stocker les données, et donc accélérer les algorithmes de classification.

Ces algorithmes sont les suivants :

⇒ **MRDIS** [79]. Cette méthode est une implémentation parallèle de l'algorithme de sélection d'instance Democratic Instance Selection (DIS). Les principaux avantages de l'algorithme DIS se révèlent être sa complexité de calcul linéaire avec le nombre d'instances, ainsi que sa structure interne, intuitivement parallélisable. MRDIS est une conception de l'algorithme DIS qui suit le modèle MapReduce; Sa mise en œuvre a été faite dans le framework de calculs distribués Spark; Elle permet une réduction du temps de traitement de manière linéaire à mesure que le nombre d'exécuteurs Spark augmente, ce qui la rend appropriée aux applications BIG DATA. La méthode démocratique pour la sélection d'instances DIS consiste à effectuer des tours d'un algorithme de sélection d'instances qui est appliqué à un certain nombre de sous-ensembles disjoints de l'ensemble de données qui constitue une partition des données disponibles. Pour chaque cycle, le processus consiste à diviser l'ensemble de données original en plusieurs sous-ensembles disjoints d'environ la même taille. Ensuite, l'algorithme de sélection d'instances est appliqué à chaque sous-ensemble séparément. Les instances à supprimer sélectionnées par l'algorithme reçoivent un vote. Ensuite, une nouvelle partition est effectuée et un autre tour de vote est effectué. Après l'exécution du nombre prédéfini de tours, les instances ayant reçu un nombre de votes supérieur à un certain seuil sont supprimées. Les auteurs ont utilisé une approche telle que la première partition est obtenue en projetant l'ensemble de

données sur un vecteur aléatoire, puis en divisant la projection en sous-ensembles de taille égale. La procédure est répétée pour obtenir les partitions suivantes. Cette façon de partitionner est spécialement conçue pour les algorithmes de sélection d'instance k-NN.

⇒ **MRVIS** [80]. Il s'agit d'un algorithme de sélection d'instances basé sur un mécanisme de vote, proposé avec MapReduce et le classificateur random weights networks (RWN). Tout d'abord, pendant la phase map, l'algorithme partitionne les grands ensembles de données en quelques petits sous-ensembles et les déploie sur différents nœuds de cloud computing. Deuxièmement, les instances informatives sont sélectionnées en parallèle avec un algorithme de sélection d'instances les auteurs ont utilisé l'algorithme Condensed Nearest Neighbours (CNN) qui lui-même utilise k-NN. Troisièmement, le reduce est appliqué pour collecter les instances sélectionnées à partir de différents nœuds de cloud computing et un sous-ensemble d'instances sélectionné est obtenu. Les trois processus précédents sont répétés p fois (p est un paramètre défini par l'utilisateur), et des sous-ensembles d'instances p sont obtenus. Enfin, la méthode de vote est utilisée pour sélectionner les instances les plus informatives des sous-ensembles p. Le classifieur random weights networks (RWN) est appliqué avec le sous-ensemble d'instances sélectionné et la précision est vérifiée sur l'ensemble de test. L'algorithme montre une performance meilleure que CNN et quelques autres approches de pointe. Cependant, dans leur étude les auteurs ont plus mis l'accent sur la précision du classifieur obtenu que sur le temps d'exécution et la scalabilité.

⇒ **MRPR** [81]. Il s'agit d'une méthode parallèle de réduction d'instances à grande échelle basée sur MapReduce qui utilise l'entropie de l'information. Elle permet une réduction de gros volumes de données sans perte de précision significative. MRPR peut traiter efficacement des ensembles de données massifs sur la plateforme Hadoop, accélérant ainsi considérablement le processus de classification et réduisant considérablement les besoins de stockage. MRPR utilise une nouvelle méthodologie de partitionnement distribuée pour les techniques de réduction de prototypes dans la classification

k-NN. Elle a été comparée à la méthode basée sur la région positive et la surpasse. Les résultats montrent que ce modèle est un outil approprié pour améliorer les performances du classificateur k-NN avec des données volumineuses. Les auteurs de MPPR ne se focalisent pas sur la précision obtenue après la sélection avec MPPR mais sur le temps d'exécution et la scalabilité.

Le tableau qui suit récapitule ces méthodes de sélection d'instances parallèles.

Algorithme parallèle	Insuffisance	Reference
MRDIS	N'est adapté qu'au classifieur k-NN	[79]
MRVIS	N'est adapté qu'au classifieur k-NN	[80]
MRPR	N'est adapté qu'au classifieur k-NN	[81]

Table III.1 - Résumé sur l'état de l'art des méthodes de sélection d'instances parallèles

Contributions :

L'inconvénient majeur des méthodes de sélection d'instances parallèles est que la plupart d'entre elles sont conçues pour le classifieur des k-plus-proches-voisins (k-NN), de sorte que les instances sélectionnées avec ces algorithmes ne sont souvent adaptées qu'aux classifieurs des k-plus-proches-voisins.

Dans ce mémoire nous proposons un tout nouveau algorithme indépendant du classifieur des k-plus-proches-voisins qui peut être appliqué à tout classifieur.

Notre méthode a présenté de très bonnes performances en terme de précision de classification et montre des bons résultats en terme de temps d'exécution et de scalabilité en comparaison avec les algorithmes de sélection d'instance de la littérature.

3. Conclusion

Dans ce chapitre nous avons présenté et discuté plusieurs algorithmes de sélection d'attributs et de sélection d'instances parallèles qui existent dans la littérature. Nous avons vu que beaucoup de ces méthodes arrivaient à donner de bons résultats avec les grands ensembles de données mais peuvent être améliorés.

Ainsi dans les chapitres qui suivent, nous allons présenter les méthodes de sélection d'attributs et de sélection d'instances parallèles que nous proposons sous l'environnement Spark afin d'améliorer ces techniques et de les rendre beaucoup plus efficientes.

Deuxième partie
Contributions

Chapitre IV

Propositions d'algorithmes de sélection d'attributs parallèles

Dans ce chapitre, nous présentons les algorithmes parallèles et évolutifs que nous avons proposés. Tous les détails concernant nos algorithmes sont donnés dans la section 2. Dans la section 3, nous évaluons nos propositions en effectuant de nombreuses expériences. Les résultats expérimentaux ont montré que nos algorithmes passent à l'échelle et permettent ainsi de traiter de manière efficace les données Big data.

1. Motivation et aperçu de la proposition

Depuis quelques décennies, la quantité de données dans le monde et nos vies semble toujours en augmentation. Aujourd'hui, les données proviennent de différentes sources, réseaux sociaux, capteurs, etc. La baisse du coût du stockage et les progrès réalisés dans la technologie de capture de données permettent aux organisations de mettre en place de très grandes bases de données, connu sous le nom BIG DATA [82]. Le traitement de ces grandes masses aide à découvrir des relations cachées et de nouvelles informations utiles [83].

Cependant ces grandes masses de données contiennent des informations redondantes ou non pertinentes qui peuvent altérer les performances des applications [82, 83]. C'est pour cette raison qu'il est primordial de les éliminer des ensembles de données. Les méthodes de sélection d'attributs sont très souvent utilisées afin d'éliminer ces données constituées d'attributs redondants ou non pertinents [1].

Mais les méthodes classiques ne sont pas capables de gérer de telles quantités de données car elles ont été conçues pour l'architecture informatique centralisée. Ils ne sont donc pas bien adaptés aux grandes quantités de données et leur efficacité peut se détériorer considérablement [1]. Ainsi, une conception parallèle et efficace des algorithmes de la sélection d'attributs s'impose pour traiter les BIG DATA [1].

Heureusement, avec la disponibilité de modèles de programmation puissants, tels que MapReduce ou Spark [1], la plupart des algorithmes de sélection d'attributs peuvent être parallélisés. Le framework Spark est une implémentation open-source de MapReduce très efficace qui a gagné en popularité [1]. De plus Spark est très adapté aux algorithmes qui fonctionnent de manière itérative comme expliqué à la section 2.3 du chapitre 1. Ce qui est le cas pour les algorithmes de sélection d'attributs. C'est pour cette raison que dans nos propositions notre choix s'est porté sur Spark.

Les expériences ont montré que nos méthodes peuvent passer à l'échelle.

2. Algorithmes de sélection d'attributs parallèles

Dans cette section, nous détaillons la logique de travail et le principe des méthodes que nous proposons. Enfin, nous introduisons l'algorithme de chaque méthode proposée et élucidons son processus de conception sous Spark.

2.1 Méthode PSF-mRMR (Parallel Scale Filter method based on mRMR)

La première méthode que nous proposons appelé PSF-mRMR (Parallel Scale Filter based on mRMR) est une version parallèle de l'algorithme de sélection d'attributs mRMR que nous avons développé sous Spark.

Soit S l'ensemble de données à traiter (avec n attributs et m instances) et K le nombre d'attributs à renvoyer. Soit $\beta \in [0, 1]$, un compromis entre le rang SVM et

le critère mRMR. x est le nombre de partitions que l'on va créer pour le jeu de données. F désigne l'ensemble des attributs du jeu de données. Soit f_i et f_j deux variables dans F. $MI(f_i, f_j)$ représente la mesure de l'information mutuelle entre f_i et f_j . $MI(l, f_i)$ représente la mesure de l'information mutuelle entre l'étiquette de classe l et l'attribut f_i.

Selon le critère mRMR :

La relevance d'un attribut f_i de F est donnée par :

$$R_{F,i} = \frac{MI(l,f_i)}{|F|} \qquad (IV.1)$$

Et la redondance de f_i est donnée par :

$$Q_{F,i} = \frac{1}{|F|^2} \sum_{f_j \in F, f_i \neq f_j} MI(f_i, f_j) \qquad (IV.2)$$

Soit ω_i le poids SVM de l'attribut f_i.

Nous proposons alors une nouvelle métrique d_i basée sur le critère mRMR mais qui prend en compte le rang SVM d'un attribut. En effet le critère SVM est connu pour ses performances en terme de capacité à sélectionner les attributs pertinents mais pas forcément ceux redondants. mRMR par contre tient compte de la redondance des attributs. Ainsi il serait intéressant de combiner la puissance des SVM avec le critère mRMR afin d'identifier au mieux les meilleurs attributs tout en évitant les attributs redondants [12]. C'est ce qui a inspiré notre méthode PSF-mRMR dans laquelle nous faisons le rapport entre le poids SVM et le critère mRMR afin d'attribuer un poids d_i à chaque attribut et choisir les attributs les plus pertinents de l'ensemble tout en éliminant ceux redondants.

Pour la ième caractéristique, le poids d_i est donnée par [12] :

$$d_i = \beta|\omega_i| + (1-\beta)\frac{R_{F,i}}{Q_{F,i}} \qquad [12] \quad (IV.3)$$

Principe de fonctionnement de PSF-mRMR.

Dans PSF-mRMR les attributs sont associés **deux à deux** dans le but de calculer l'information mutuelle entre eux, ce qui permettra de mesurer leur degré de corrélation. De même, chaque attribut est associé à l'étiquette de classe pour déterminer l'information mutuelle entre eux, ce qui permettra d'obtenir une mesure de corrélation. La valeur des informations mutuelles entre un attribut et les autres attributs de l'ensemble nous donne une idée de la redondance de l'attribut, tandis que l'information mutuelle d'un attribut avec l'étiquette de classe mesure la relevance de l'attribut à savoir sa pertinence.

PSF-mRMR affecte un score à chaque attribut en effectuant un rapport entre la relevance d'un attribut et sa redondance.

Ensuite, les attributs avec les meilleurs scores sont sélectionnés car ils sont considérés comme les plus relevants et moins redondants.

Le principe de fonctionnement de PSF-mRMR est détaillé à travers les phases suivantes :

⇒ **Dans la 1ère phase**, des sous-ensembles sont créés à partir de chaque attribut. Chaque sous ensemble a comme clé l'attribut et comme valeur un sous ensemble constitué de trois autres sous-ensembles dont le 1er représente les valeurs de l'attribut dans chaque instance, le 2ème les valeurs d'un autre attribut dans chaque instance du jeu de données et le 3ème contient les valeurs des étiquettes de classe prises au niveau de chaque instance. Par exemple, si a_i est un attribut, $\{ c^1_i, c^2_i, .., c^m_i \}$ les différentes valeurs de a_i pris dans chaque instance , $\{ c^1_j, c^2_j .. c^m_j \}$ les valeurs de l'attribut a_j pris dans chaque instance, $\{l_1, l_2, \cdots, l_m\}$ les valeurs des étiquettes de classe prises dans les m instances. Alors a_i et a_j sont associés à travers l'ensemble $\{ a_i, \{ c^1_i, c^2_{i,..} c^m_i \}, \{ c^1_j, c^2_j .. c^m_j \}, \{l_1, l_2, \cdots, l_m\} \}$.

Pour obtenir ces ensembles, les attributs sont au départ distribués entre les workers en créant x partitions de F et en envoyant chaque partition à un worker. Au niveau de chaque worker, chaque attribut a_i pris en charge par le worker est associé à chaque autre attribut a_j du jeu de données ainsi qu'aux valeurs des étiquettes de classe comme expliqué précédemment. Ce qui donne plusieurs ensembles pour chaque attribut, puisque chaque attribut associé à un autre donne un ensemble.

La figure **IV.1** illustre cette phase.

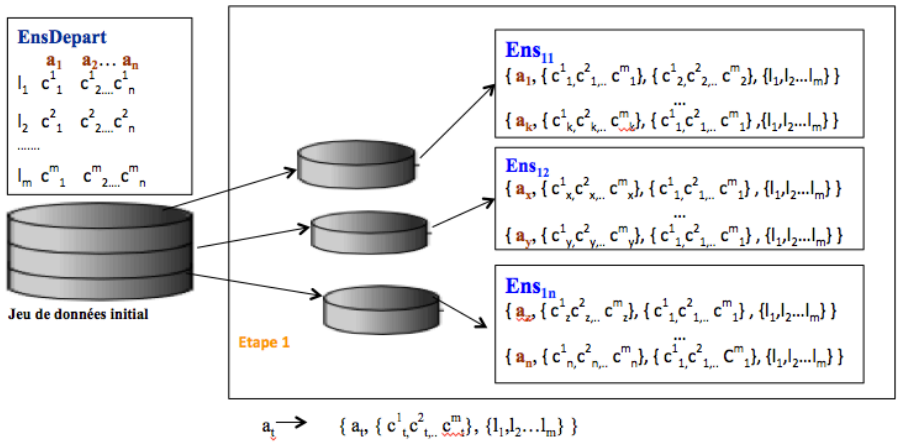

Figure IV.1 - Distribution des attributs entres les nœuds

⇒ **Dans la 2ème phase**, chaque ensemble de la forme $\{ a_i, \{ c^1_i, c^2_{i,..} c^m_i\}, \{c^1_j, c^2_j, \cdots, c^m_j\}, \{l_1, l_2, \cdots, l_m\} \}$ est mappé à un ensemble de la forme $\{ a_i, \{mij\}, \{mic\} \}$.

mij est l'information mutuelle entre l'attribut a_i et l'attribut a_j.

mic représente l'information mutuelle entre l'attribut a_i et l'étiquette de classe.

Les ensembles $\{c^1_i, c^2_i, \cdots, c^m_i\}$ et $\{c^1_j, c^2_j, \cdots, c^m_j\}$ ont permis de calculer mij; $\{c^1_i, c^2_i, \cdots, c^m_i\}$ et $\{l_1, l_2, \cdots, l_m\}$ ont permis d'obtenir mic.

Cette phase est illustrée à la figure **IV.2**.

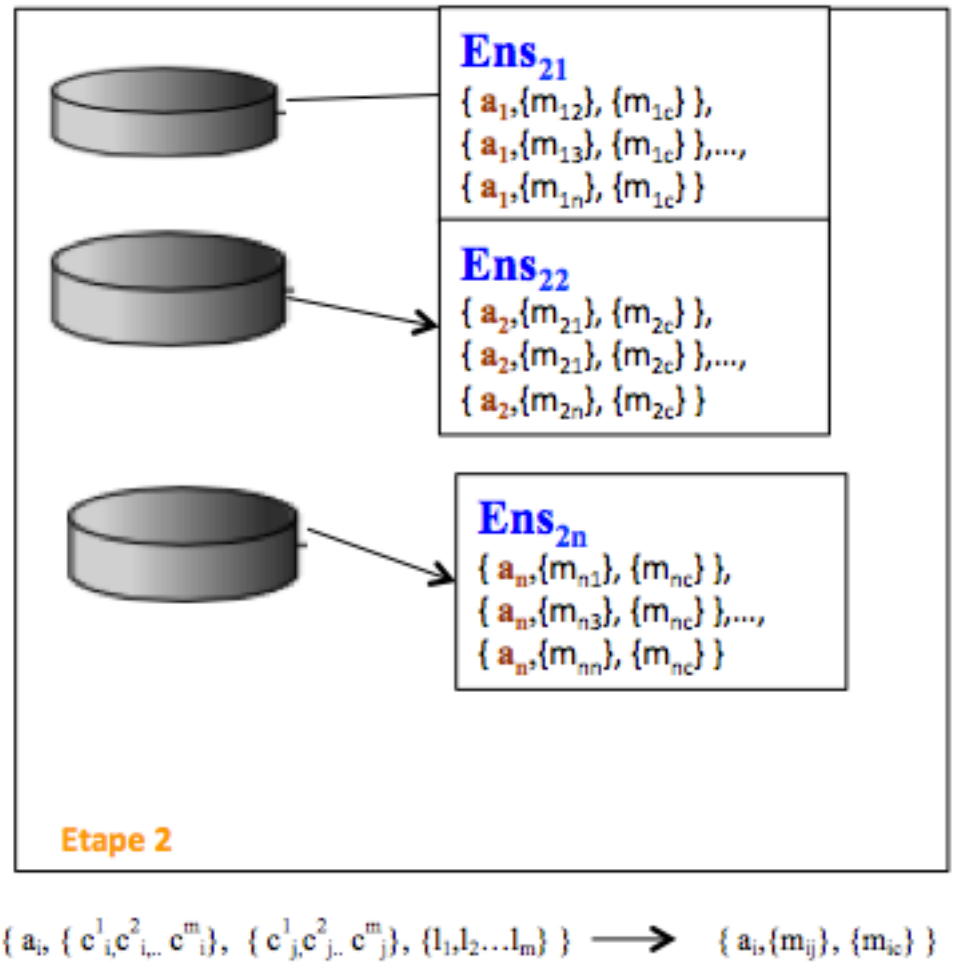

Figure IV.2 - Calcul de l'information mutuelle entre les attributs et la classe et entres les attributs eux mêmes

⇒ **Dans la 3eme phase,** la somme des informations mutuelles entre chaque attribut et les autres attributs de l'ensemble est calculée dans le but d'obtenir la redondance d'un attribut.

Donc pour chaque attribut a_i, on aura un ensemble de la forme $\{a_i, \sum_{j=1}^{n} mij, mic\}$, obtenu en agrégeant les informations mutuelles de a_i obtenus au niveau des sous ensembles.

Ce qui signifie que chaque attribut a_i sera associé à la somme de son information mutuelle $\sum_{j=1}^{n} mij$, avec chaque autre attribut du jeu de données et son information mutuelle mic avec la classe.

Ce principe est montré à la figure **IV.3**.

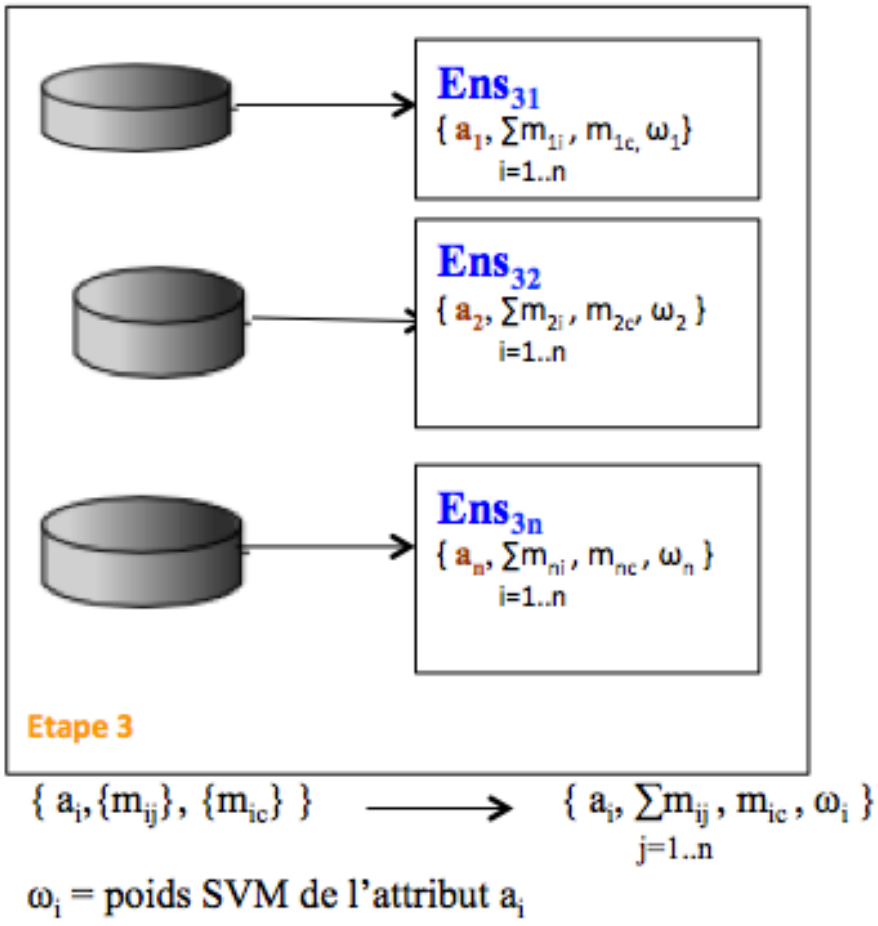

Figure IV.3 - Somme des informations mutuelles entre attributs

⇒ **Dans la 4ème étape**, le score final d_i de chaque attribut a_i est calculé à partir de la somme de son information mutuelle avec les autres attributs et son information mutuelle avec la classe.

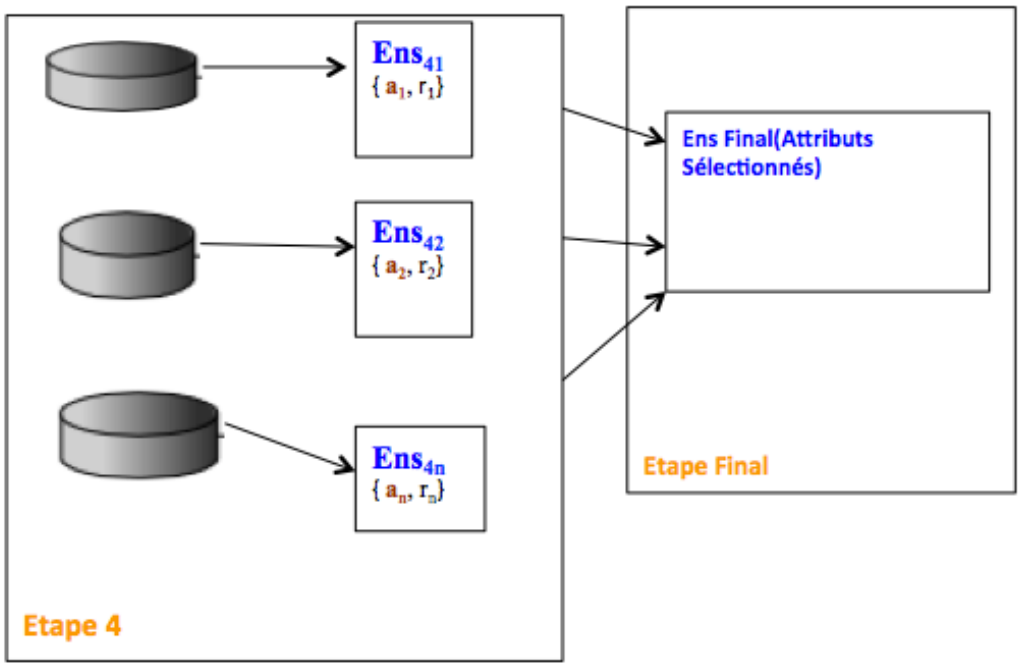

Figure **IV.4 -** Calcul du critère mRMR pour chaque attribut.

Pour finir, les scores d_i sont envoyés au worker qui les ordonne et sélectionne les K attributs ayant obtenus les plus grandes valeurs pour d_i.

Ces différentes phases sont résumées à travers l'algorithme suivant :

⇒ **Étape 1: créer x partitions d'attributs**

1. Construire l'ensemble labels = $\{l_1,..,l_m\}$ l'ensemble constitué des étiquettes de classe dans chaque instance du jeu de données.
2. Construire *values* = $\{\{v_i^1,..,v_i^m\}$, i=1 to n$\}$

   values représente l'ensemble des valeurs v_i^j de chaque attribut f_i dans chaque instance I_j:
3. Construire x sous-espaces de caractéristiques SF_t, $t = 1..x$ à partir de l'espace des caractéristiques F.
4. Construire x sous-espaces sub_t de $\{\{v_i^1,..,v_i^m\},\ i \in SF\}$

   Chaque ensemble sub_t sera envoyé à un unique worker (parmi les x worker).

⇒ **Étape 2 : associer les attributs par deux avec les étiquettes de classe**

Au niveau de chaque worker t:

5. Créer plusieurs ensembles pour chaque attribut f_i dans sub_t en mappant f_i avec chaque autre attribut f_j de F comme suit:

$$f_i => \{f_i, \{v_i^1,..,v_i^m\}, \{v_j^1,..,v_j^m\}, \{l_1,..,l_m\}\}$$

Nous appelons l'ensemble $\{f_i, \{v_i^1,..,v_i^m\}, \{v_j^1,..,v_j^m\}, \{l_1,..,l_m\}$, i=1 to n, j=1 to m} obtenu : *rdd2*.

⇒ **Étape 3 : calculer l'information mutuelle entre les attributs et la pertinence de chaque attribut**

6. Dans cette étape, nous utilisons chaque élément de rdd2 pour calculer l'information mutuelle entre chaque caractéristique f_i et chaque autre caractéristique f_j de F. Nous calculons aussi la pertinence R_i (information mutuelle avec son étiquette de classe) de f_i. Nous procédons comme suit :

Foreach element $e \subset rdd2$

rdd $[(f_i, \mathrm{M}_{ij}, R_i)]$ = *mapToPair* $(e=>\{f_i, \mathrm{M}_{ij}, R_i\})$

M_{ij} = *MutualInformation* $(\{v_i^1,..,v_i^m\}, \{v_j^1,..,v_j^m\})$

R_i= *MutualInformation* $(\{v_i^1,..,v_i^m\}, \{l_1,..,l_m\})$ */n*

$l_{k\in 1..m}$ *représente l'étiquette de classe dans l'instance* I_k.

EndForeach

Les ensembles constitués de chaque caractéristique f_i, son information mutuelle avec une autre caractéristique f_j et son information mutuelle R_i avec l'étiquette de classe seront appelées rdd3.

⇒ **Étape 4 : agréger les informations mutuelles pour chaque attribut en les additionnant.**

7. Afin d'obtenir la redondance de chaque caractéristique f_i, additionner ses informations mutuelles avec les autres caractéristiques de F et conserver l'information mutuelle avec l'étiquette de classe (pour la pertinence). Un nouvel ensemble est alors obtenu et nous l'appelons rdd4. Chaque élément de rdd4 se composé de $\{f_i, sumM_{ij}, R_i\}$, f_i est la

caractéristique, $sumM_{ij}$ est la somme des informations mutuelles entre f_i et les autres attributs de l'espace des caractéristiques F et R_i l'information mutuelle entre f_i et l'étiquette de classe.

Cela correspond aux instructions suivantes :

Foreach element (f_i, M_{ij}, R_i) $\in$ *rdd3*

rdd [(f_i, $sumM_{ij}$, R_i)] = *reduceByKey* (_+_)

$$sumM_{ij} = \sum_{i=1}^{n} M_{ij}$$

EndForeach

⇒ **Étape 5 : pour chaque attribut, calculer le poids SVM**

8. Dans cette étape, calculer pour chaque caractéristique f_i son poids SVM ω_i comme suit:

Foreach $f_i \subset F$

rdd [(f_i, ω_i)] = *map* (f_i => {f_i, ω_i})

$\omega_i \subset \omega$, *where* ω = *SVMWeight (F)*

EndForeach

⇒ **Étape 6 : pour chaque attribut, calculer le rapport entre sa pertinence et sa redondance**

9. Dans cette étape, calculer pour chaque caractéristique f_i son rang d_i qui représente un compromis entre la redondance et la pertinence de f_i.
10. Ensuite, envoyer toutes les valeurs d_i au nœud maître.

Cela se fait comme suit :

Foreach element (f_i, $sumM_{ij}$, R_i) $\subset$ *rdd4*

rdd [(f_i, d_i)] = *mapToPair* ({f_i, $sumM_{ij}$, R_i} => {f_i, d_i})

Q_i = $sumM_{ij}$ / (n*n);

d_i = β + ω_i + ((1-β)* (R_i / Q_i));

/ ω_i *is the SVM weight of attribute* f_i*/*

EndForeach

All workers send d_i *to the master*

⇒ **Étape 7 : Choisissez les meilleurs attributs de F.**

11. Enfin, le nœud maître recueille, ordonne et renvoie les K attributs avec les meilleurs scores d_i. Cela correspond aux instructions suivantes :

On the master :

Collect and take ordered

Return S': optimal subset of K features in S with highest scores d_i.

2.2 La méthode de sélection d'attributs parallèle PNFS_Spark (Parallelized Neighbor Feature Selection_Spark)

Avec l'algorithme PNFS_Spark, nous proposons une toute nouvelle métrique basée sur l'utilisation de la valeur médiane de la pertinence des attributs avec l'étiquette de classe. La médiane donne une idée satisfaisante de la tendance générale d'une série statistique (l'ensemble des attributs F ici).

PNFS_Spark calcule au départ la relevance de chaque attribut puis détermine la médiane de ces relevances que nous appelons R_{fe}.

Par la suite, pour chaque attribut, ses voisins sont déterminés. Un attribut f_i est considéré comme le voisin d'un autre attribut f_j si l'information mutuelle M_{ij} entre les 2 est supérieure à la valeur médiane R_{fe}. En effet nous considérons que cela indique une forte corrélation entre les deux attributs f_i et f_j. Si nV est le nombre de voisins d'un attribut f_i et R_i sa relevance, alors le score final d_i de l'attribut f_i sera : $nV*R_i$. Nous considérons que l'attribut est d'autant plus pertinent qu'il a de voisins. C'est pour cette raison que sa relevance est multipliée par son nombre de voisins. Donc plus

il a de voisins plus son importance augmente. Les attributs qui ont obtenu les meilleurs scores sont renvoyés par notre algorithme.

PNFS_Spark suit sept étapes :

⇒ **Étape 1: créer *x* partitions d'attributs**

1. Construire l'ensemble labels = $\{l_1,..,l_m\}$ constitué des étiquettes de classe des m instances du jeu de données.
2. Construire *values* = $\{\{v_i^1,..,v_i^m\}$, i=1 to n$\}$

   values représente l'ensemble des valeurs v_i^j de chaque attribut f_i dans chaque instance I_j:
3. Construire x sous-espaces de caractéristiques SF_t, $t = 1..x$ à partir de l'espace des caractéristiques *F*.
4. Construire x sous-espaces sub_t de $\{\{v_i^1,..,v_i^m\},\ i \in SF\}$
5. Chaque sub_t sera envoyé à un unique worker (parmi les x worker).

⇒ **Étape 2 : associer les attributs par deux avec les étiquettes de classe**

Au niveau de chaque worker t :

6. Créer plusieurs ensembles pour chaque attribut f_i dans sub_t en mappant f_i avec chaque autre attribut f_j de F comme suit :
   1. $f_i => \{f_i, \{v_i^1,..,v_i^m\}, \{v_j^1,..,v_j^m\}, \{l_1,..,l_m\}\}$

Nous appelons l'ensemble $\{f_i, \{v_i^1,..,v_i^m\}, \{v_j^1,..,v_j^m\}, \{l_1,..,l_m\}$, i=1 to n, j=1 to m$\}$ obtenu *rdd2*.

⇒ **Étape 3 : calculer l'information mutuelle entre les attributs et la pertinence de chaque attribut**

7. Dans cette étape, nous utilisons chaque élément de rdd2 pour calculer l'information mutuelle entre chaque caractéristique f_i et une autre caractéristique f_j de F. Nous calculons aussi la pertinence R_i (information mutuelle avec son étiquette de classe) de f_i. Nous procédons comme suit :

Foreach element $e \subset rdd2$

$rdd\ [(f_i, \mathrm{M}_{ij}, R_i)] = mapToPair\ (e=>\{f_i,\ \mathrm{M}_{ij}, R_i\})$

$\mathrm{M}_{ij} = MutualInformation\ (\{v_i^1,..,v_i^m\},\ \{v_j^1,..,v_j^m\})$

$R_i = MutualInformation\ (\{v_i^1,..,v_i^m\},\ \{l_1,..,l_m\})\ /n$

$l_{k\in 1..m}$ *représente l'étiquette de classe dans l'instance* I_k.

EndForeach

Les ensembles constitués de chaque caractéristique f_i, son information mutuelle avec une autre caractéristique f_j et son information mutuelle R_i avec l'étiquette de classe forment l'ensemble que nous appelons rdd3.

⇒ **Étape 4 : calculer la valeur médiane de la pertinence des attributs avec l'étiquette de classe.**

Calculez $\boldsymbol{R_{fe}}$ la valeur médiane de l'ensemble des valeurs de pertinence R_i.

Soit f_e, l'indice de la valeur médiane de la liste des valeurs de pertinence entre les caractéristiques et les étiquettes de classe. Si n est pair alors f_e = (n+1)/2 sinon f_e = (n/2 +n/2+1)/2.

8. If (n%2==0)

   f_e = (n+1)/2

   else

   f_e = (n/2 +n/2+1)/2

   R_{fe}= *rdd3* [f_e]. R_e

⇒ **Étape 5 : pour chaque attribut, calculer le poids SVM**

9. Dans cette étape, déterminer pour chaque attribut f_i, l'ensemble $Vois_{ij}$ constitué de ses voisins.

   Cela se fait comme suit :

   On each worker x:

   Foreach $f_i \subset \mathrm{sub_x}$

   Foreach $f_j \subset \mathrm{F}$

   / M_{ij} represents the mutual information between f_i and f_j */*

   If(M_{ij})>R_{fe}

$rdd\ [(f_i, Vois_{ij})] = mapToPair\ (\{f_i\} => \{f_i, f_j\})$

EndForeach

EndForeach

All workers send d_i *to the master*

⇒ **Étape 6 : pour chaque attribut, calculer son score**

10. Dans cette étape, calculer pour chaque attribut f_i, la mesure de classement d_i qui représente le score final de l'attribut.
11. Ensuite, envoyer toutes les valeurs au maître.

Cela se fait comme suit :

Foreach element $f_i \subset rdd\ [(f_i, Vois_i)]$

Foreach element $(f_j \subset rdd\ [f_i]\ .Vois_i)$

$rdd\ [(f_i, d_i)] = mapToPair\ (\{f_i\} => \{f_i, d_i\})$

/ determines nV number of neighbor of* f_i **/*

$nV = Vois_i.length$

/ the score* d_i *of* f_i *is obtained by multiplying its relevance* R_i *with its number of neighbor nV */*

$$d_i = \sum_{i=1}^{nV} R_i$$

EndForeach

EndForeach

All workers send r_i *to the master*

⇒ **Étape 7 : Choisir les meilleurs attributs de F.**

12. *Enfin, le nœud maître recueille, ordonne et renvoie les K attributs avec les meilleurs scores* d_i.

Cela correspond aux instructions suivantes :

On the master:

Collect and take ordered

Puisque PNFS_Spark n'a pas d'équivalent centralisé, nous avons créé une méthode **CNFS_Spark** qui représente PNFS_Spark mais qui s'exécute sur une seule machine. **CNFS_Spark** utilise les mêmes métriques que PNFS_Spark.

2.3 La méthode de sélection d'attributs parallèle PS-RFE-SVM

Récemment, RFE-SVM a montré d'excellentes performances en matière de classification et de prédiction, et est largement utilisé pour le diagnostic des maladies ou l'assistance médicale. Cependant, le principal obstacle à son utilisation sur les données Big data, est la puissance de calcul requise. RFE-SVM doit donc être adapté à la distribution et à la parallélisation.

Ainsi nous proposons une version parallèle de l'algorithme de sélection d'attributs RFE-SVM sous spark que nous appelons PS-RFE-SVM.

Soit *S* l'ensemble de données en entrée (avec *n* caractéristiques et *m* instances) et *K* le nombre d'itérations à effectuer. Soit x le nombre de partitions pour l'ensemble de données. *F* désigne l'espace des caractéristiques. La sortie *S'* sera le sous-ensemble optimal de *F* avec les meilleures caractéristiques choisies par notre algorithme.

Notre algorithme suit 5 étapes:

⇒ **Etape 1: créer x partitions d'attributs**

1. Construire *labels* = $\{l_1,..,l_m\}$ l'ensemble constitué des étiquettes de classe de chaque instance de l'ensemble de données.
2. Construire *values* = $\{\{v_i^1,..,v_i^m\}$, i=1 to n$\}$

   values représente l'ensemble des valeurs v_i^j de l' attribut f_i dans l'instance I_j:
3. Construire x sous-espaces d'attributs SF_t, t = 1..x à partir de l'espace de caractéristiques F.
4. Construire x sous-espaces sub_t de $\{\{v_i^1,..,v_i^m\},\ i \in SF\}$.
5. Chaque ensemble sub_t sera envoyé à un unique worker (parmi les *x* workers).

⇒ **Étape 2 : entrainer le SVM sur l'ensemble de données d'entrée**

6. Entrainer un SVM sur l'ensemble d'apprentissage (l'ensemble de données d'entrée S); ce qui nous donne un modèle ω qui représente l'ensemble des poids SVM de chaque caractéristique de F.

⇒ **Etape 3 : Associer chaque attribut à son poids SVM**

7. Dans cette étape, créez un rdd constitué de chaque caractéristique f_i de F et de son poids SVM ω_i comme suit:

On each worker t:

Foreach $f_i \subset SF_t$

$rdd\ [(f_i,\ \omega_i)] = map\ (f_i => \{f_i, \omega_i\})$

$\omega_i \subset \omega$, *where* ω *=SVMWeight (F)*

EndForeach

All workers send ω_i *to the master*

⇒ **Etape 4 : Éliminer l'attribut avec le plus petit poids ;**

8. Enfin, le nœud maître recueille, ordonne et élimine l'attribut avec le plus petit poids. Cela correspond aux instructions suivantes :

On the master:

Collect and take ordered

Return S': optimal subset of K features in S with highest scores ω_i .

⇒ **Etape 5 :** Répétez K fois le processus de l'étape 2 à l'étape 4 avec le jeu de données restreint aux attributs restantes.

A noter qu'il est possible d'éliminer plusieurs attributs pendant une itération.

3. Expériences

3.1 Description des données

Pour réaliser nos expériences, nous avons utilisé des jeux de données réels de référence, choisis sur le site mldata.org [84]. Ces jeux de données sont présentés dans la table **IV.1**. L'objectif est de comparer nos méthodes parallèles (PSF-mRMR, PNFS_Spark et PS-RFE-SVM) à leurs équivalents classiques.

Pour PS-RFE-SVM en particulier, en plus des expériences effectuées avec ces jeux de données, d'autres expériences ont été faites avec les jeux de données de la table **IV.2** toujours dans le but de démontrer son efficience avec les grands ensembles de données.

Nous avons utilisé le classifieur SVM pour la classification et les données utilisées sont au format LibSVM.

Nom	**Nombre d'attributs**	**Nombre d'instances**
Colon-cancer	2000	62
Colon-Tumor	2000	60

Table IV.1- Caractéristiques des ensembles de données de référence pour la 1ère expérience.

Les expériences ont d'abord été réalisées sur un cluster composé de 4 nœuds, chaque nœud ayant 8 cœurs fonctionnant à 2,60 GHz, avec 56 Go de mémoire et un disque de 382 Go, puis sur un cluster de 6 nœuds avec les mêmes paramètres. Les nœuds de calcul s'exécutent tous sur linux. Dans les expériences qui consistent à comparer les méthodes parallèles et les méthodes centralisés, nous avons utilisé des ensembles de données limités à 2000 caractéristiques, car au-delà de ce nombre, les méthodes centralisées prennent trop de temps à s'exécuter. Par exemple, pour certains jeux de données de taille supérieure à 2000, les méthodes centralisées peuvent prendre plusieurs jours pour s'exécuter complètement.

Nom	Nombre d'attributs	Nombre d'instances
DUKE	7129	86
OVARIAN	15154	253
BREAST	24481	97

Table IV.2- Caractéristiques des ensembles de données de référence la 2ème expérience.

3.2 Résultats

3.2.1 PSF-mRMR

⇒ 1ere expérience.

Pour l'évaluation de la précision de la classification de notre proposition, le tableau **IV.3** et la figure **IV.5** illustrent les résultats obtenus par notre algorithme en sélectionnant différents pourcentages de l'ensemble original des caractéristiques. Précision du classifieur avec les jeux de données de référence.

Pourcentage d'attributs sélectionnés du jeu de données	Précision du classifieur		
	BREAST	DUKE	OVARIAN
100%	0,7526	0,5227	0,7549
75%	0,6598	0,9773	0,8221
50%	0,7629	0,9773	0,4744
25%	0,866	0,9773	0,8537

Table IV.3- Précision du classifieur avec les jeux de données de référence.

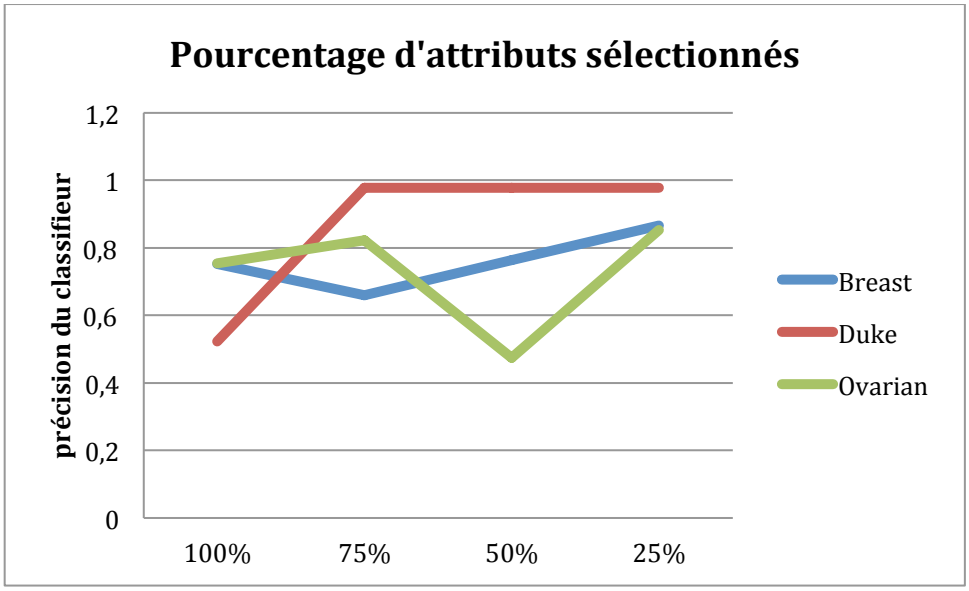

Figure IV.5 - Précision du classifieur pour les jeux de données.

Comme nous pouvons le remarquer les précisions s'améliorent la majeure partie du temps avec la réduction des données.

Ce qui est particulièrement remarquable est que, pour tous les ensembles de données, la précision de la classification est bien meilleure pour un sous-ensemble de 25% des caractéristiques. Ainsi avec 75% des caractéristiques enlevés du jeu de données, de biens meilleures performances sont obtenus. Ce qui montre la capacité de notre méthode à réduire considérablement la taille des données tout en améliorant la précision.

Après avoir discuté de la performance de notre proposition en termes de précision de classification, nous avons également étudié son évolutivité (scalabilité).

Pour ce faire, nous avons varié le nombre de cœurs, et effectué tous les tests sous les mêmes conditions.

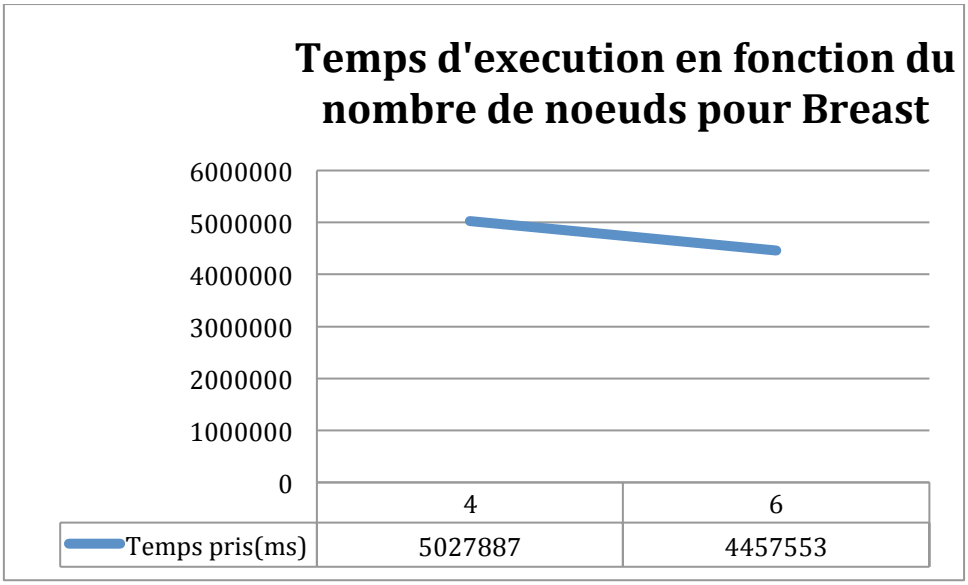

Figure IV.6 - Évolutivité du jeu de données Breast.

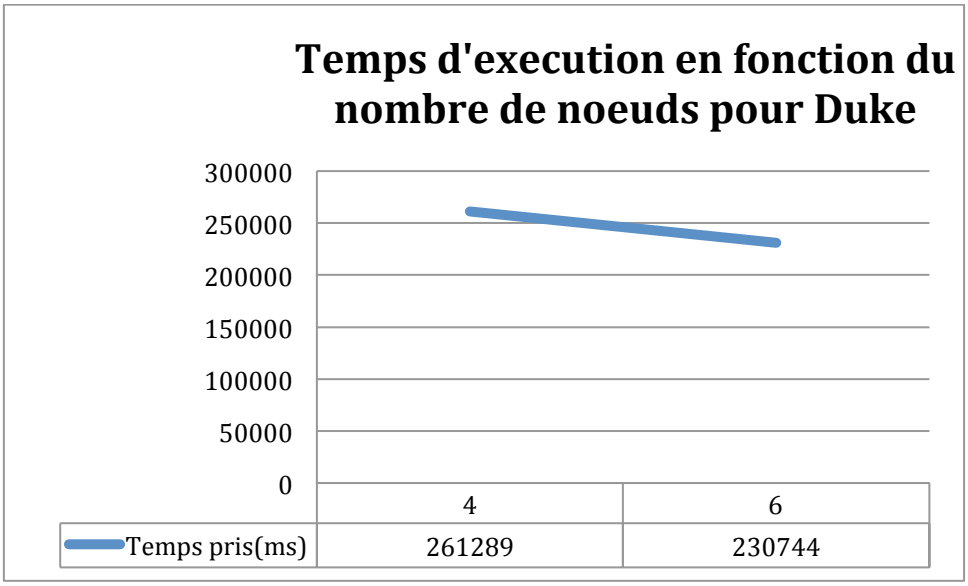

Figure IV.7 - Évolutivité du jeu de données Duke.

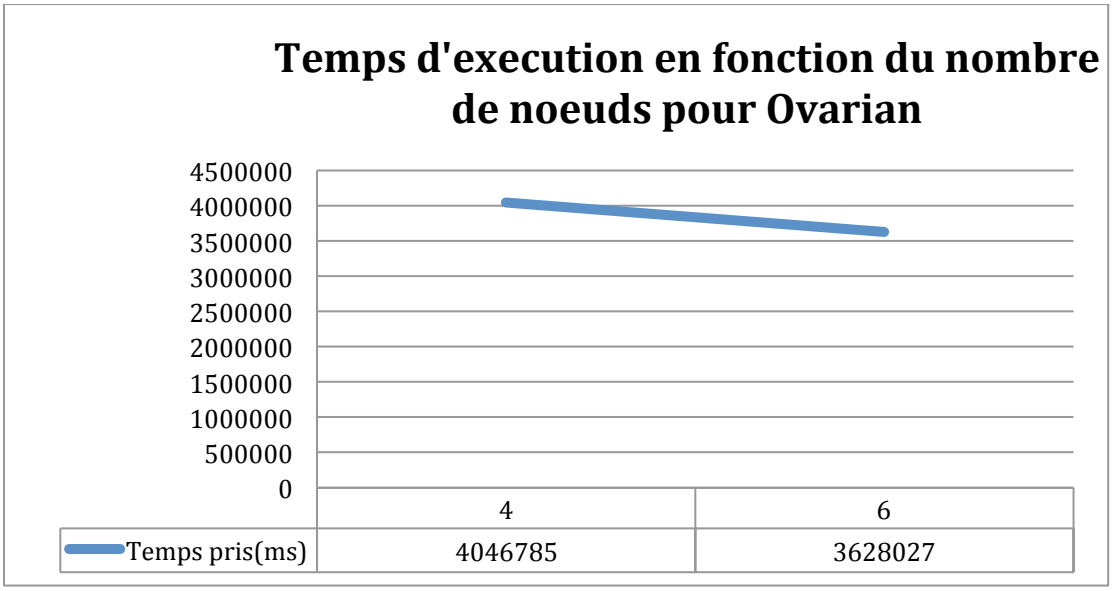

Figure IV.8 - Évolutivité du jeu de données Ovarian.

Les résultats démontrent que notre solution offre une bonne efficacité de calcul. Le temps de sélection d'attributs diminue de manière significative pour la majorité des jeux de données lorsque le nombre de nœuds augmente.

⇒ 2ème expérience

Dans la 2ème expérience, nous discutons d'abord de l'évolutivité de notre solution, puis nous comparerons le temps d'exécution de notre proposition avec celui de la méthode mRMR centralisé.

Les figures **IV.9**, **IV.10** et **IV.11** montrent respectivement comment le temps d'exécution varie en fonction du nombre de nœuds lorsque nous sélectionnons 25%, 50% ou 75% de l'ensemble de données colon-cancer.

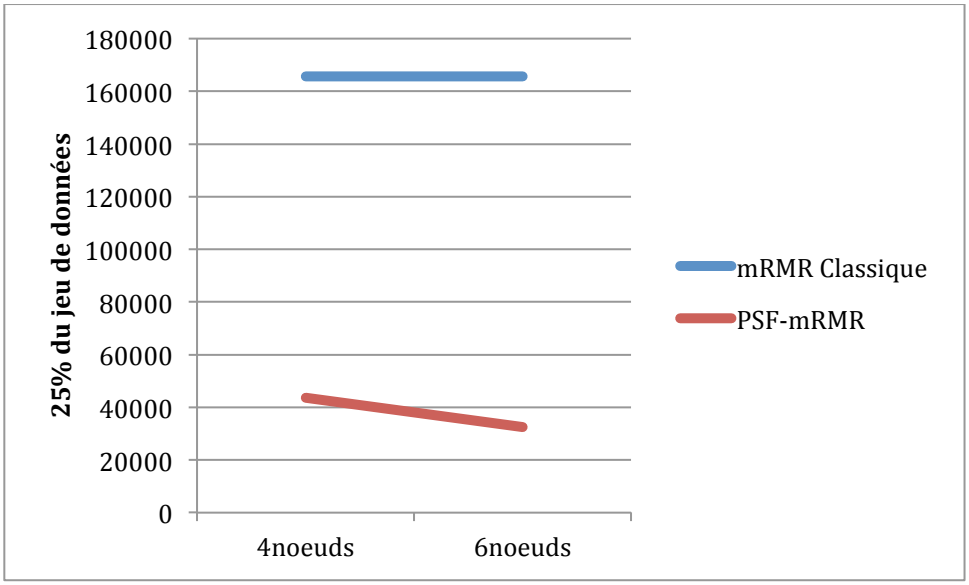

Figure IV.9 - Évolutivité de PSF-mRMR et mRMR classique avec 25%.

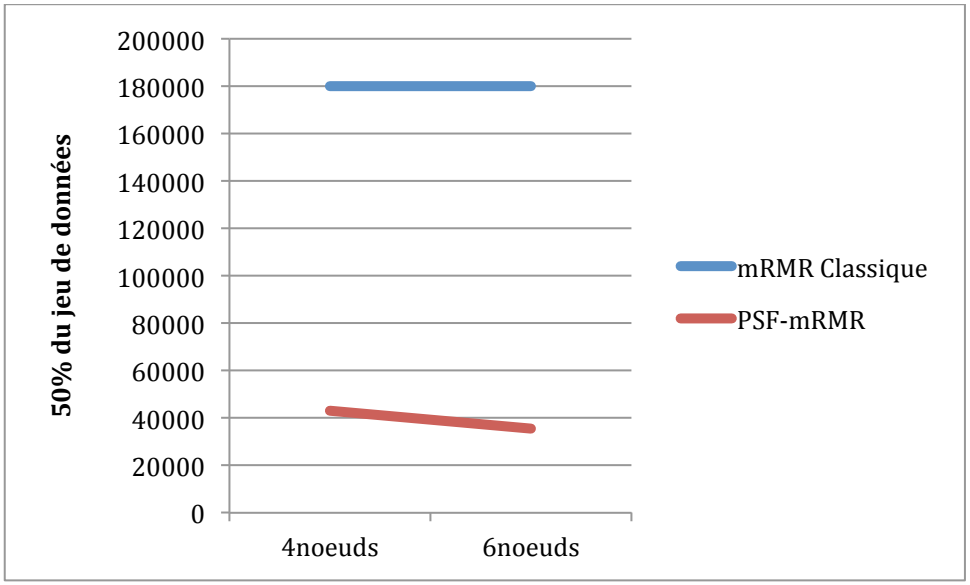

Figure IV.10 - Évolutivité de PSF-mRMR et mRMR classique avec 50%.

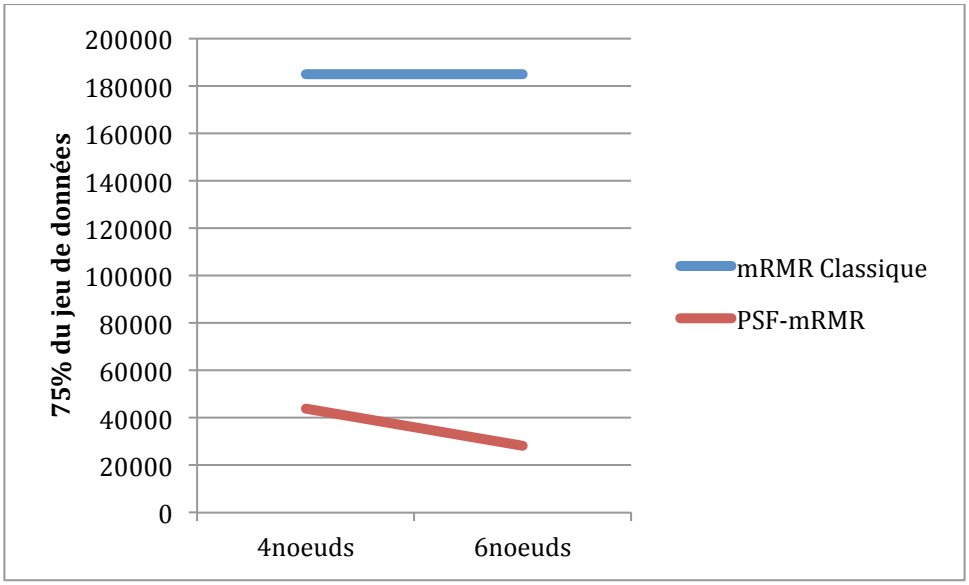

Figure IV.11 - Évolutivité de PSF-mRMR et mRMR classique avec 75%.

Les figures montrent clairement que le temps d'exécution de notre proposition décroit considérablement lorsque le nombre de nœuds augmente alors que le temps pris par la méthode MRMR classique reste constant.

Nous avons également étudié le temps d'execution de notre méthode en comparaison avec mRMR classique et de la méthode fast-mRMR présenté dans l'état de l'art des méthodes parallèles. Nous présentons ici d'abord les résultats pour colon-cancer ensuite colon-tumor.

- ✓ **Colon-cancer**

Les figure **IV.12** et **IV.13** montrent le temps pris par notre méthode en comparaison à celui de la méthode mRMR classique et de la méthode fast-mRMR présenté dans l'état de l'art des méthodes parallèles, avec respectivement 4 et 6 nœuds.

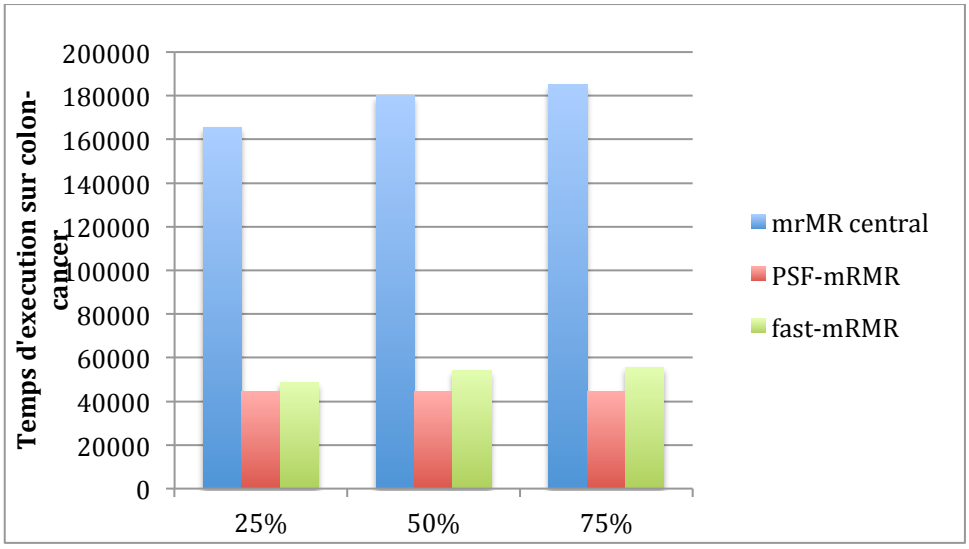

Figure IV.12 - Temps pris par colon-cancer avec 4noeuds.

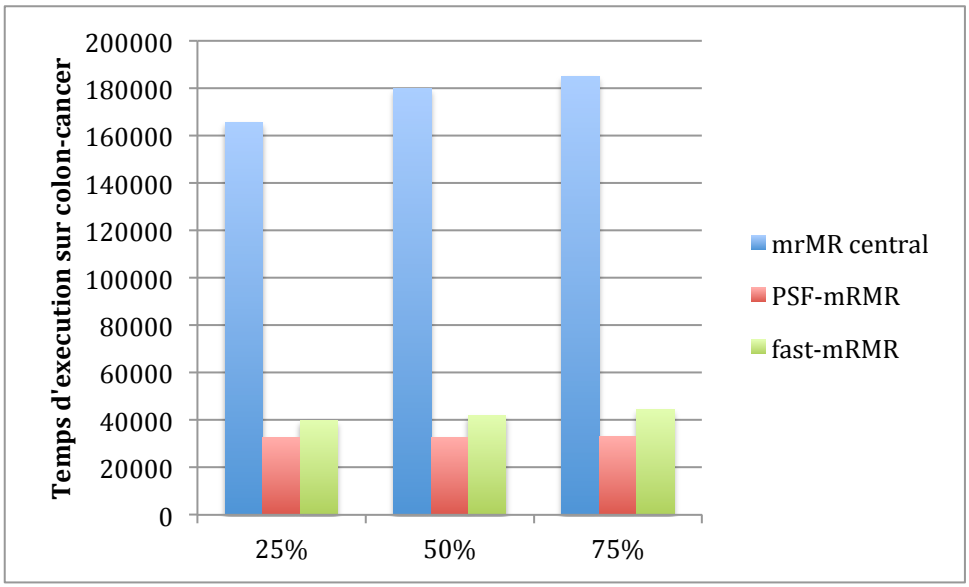

Figure IV.13 - Temps pris par colon-cancer avec 6 noeuds.

Nous avons mesuré la métrique d'accélération (speed-up) en évaluant de combien est ce que l'algorithme **PSF-mRMR** est plus rapide que l'implémentation **mRMR classique** que l'on a appelé ici mRMR central.

Nous mesurons la métrique d'accélération avec la formule suivante [7] :

$$S = \frac{T_{mRMR}}{T_{PSF-mRMR}} \qquad (IV.1)$$

Où S est le speed-up résultant, $\boldsymbol{T_{PSF-mRMR}}$ est le temps d'exécution de PSF-

mRMR et $\boldsymbol{T_{mRMR}}$ est le temps d'exécution de mRMR.

En moyenne la valeur du speed-up varie entre 4 et 5 pour notre méthode. Comme on peut le constater donc, le temps d'exécution de notre méthode est au moins 4 fois plus petit que celui de la méthode mRMR classique.

Nous avons aussi mesuré la métrique d'accélération (speed-up) en évaluant de combien est ce que l'algorithme **PSF-mRMR** est plus rapide que l'implémentation **fast-mRMR** présenté dans l' état de l'art des méthodes de sélection d'attributs.

En moyenne la valeur du speed-up varie entre 1 et 2 pour notre méthode. Ce qui veut dire que le temps d'exécution de notre méthode est 1 à 2 fois plus petit que celui de la méthode fast-mRMR.

✓ **Colon-Tumor**

Pour colon-tumor, les résultats obtenus avec 4 nœuds sont ceux donnés à la figure **IV.14**:

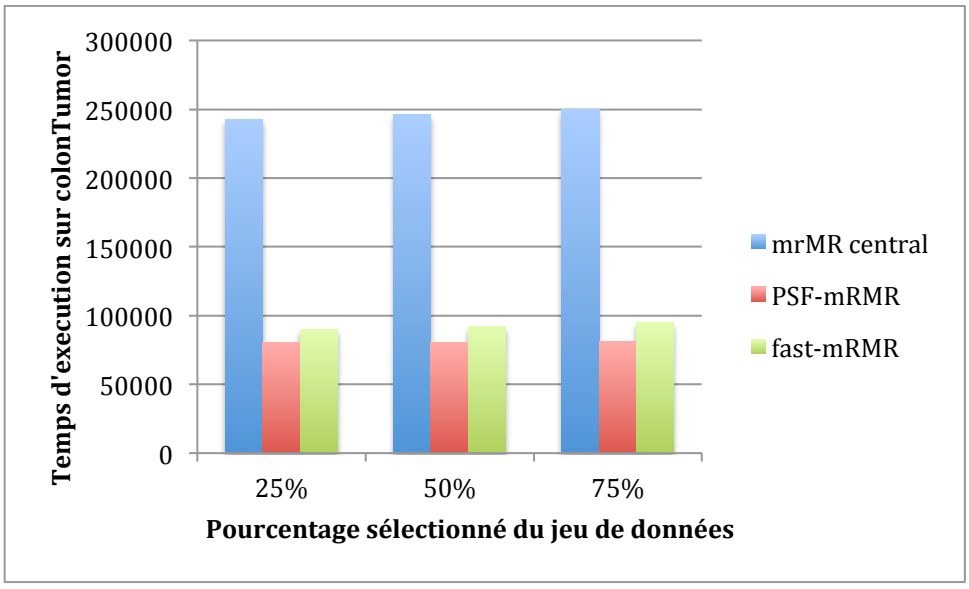

Figure IV.14 - Temps pris par colon-tumor avec 4noeuds.

Avec un cluster de 6 nœuds, le temps d'exécution est indiqué sur la figure **IV.15**.

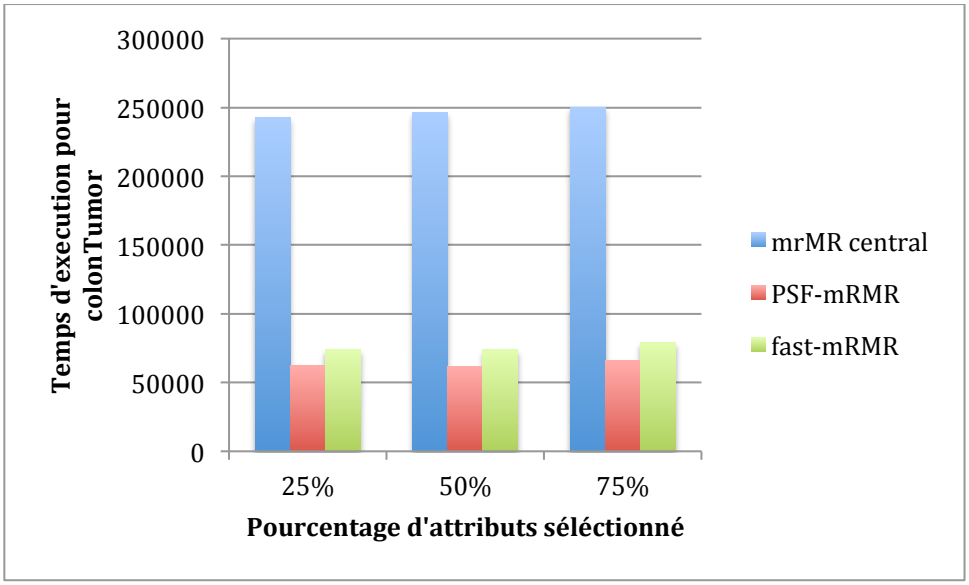

Figure IV.15 - Temps pris par colon-tumor avec 6 noeuds.

Ici aussi nous avons mesuré la métrique d'accélération (speed-up) pour savoir de combien est ce que l'algorithme PSF-mRMR est plus rapide que l'implémentation mRMR.

Nous mesurons la métrique d'accélération avec la formule suivante :

$$S = \frac{T_{mRMR}}{T_{PSF-mRMR}} \quad [7]$$

Où S est le speed-up résultant, $T_{PSF-mRMR}$ est le temps d'exécution de PSF-mRMR et T_{mRMR} est le temps d'exécution de mRMR.

En moyenne la valeur du speed-up varie ici entre 4 et 5 pour notre méthode.

De la même façon qu'avec colon-cancer, nous pouvons remarquer qu'avec colon-tumor aussi, le temps d'exécution de notre méthode parallèle est également au moins 4 fois plus petit que celui de la méthode mRMR classique. De plus le temps d'exécution de notre méthode PSF-mRMR est également 1 à 2 fois plus petit que celui de la méthode fast-mRMR.

Par conséquent, nous pouvons conclure de ces expériences que notre solution surpasse la méthode mRMR classique et la méthode fast-mRMR en termes de temps d'exécution. De plus, plus on augmente le nombre de nœuds, plus le temps d'exécution diminue pour notre méthode alors que celui de mRMR classique reste constant.

3.2.2 PNFS_Spark

Les figures **IV.16**, **IV.17** et **IV.18** montrent respectivement la variation du temps d'exécution de PNFS_Spark et son équivalent central CNFS_Spark en fonction du nombre de nœuds, lorsque l'on sélectionne 25%, 50% ou 75% de l'ensemble de données.

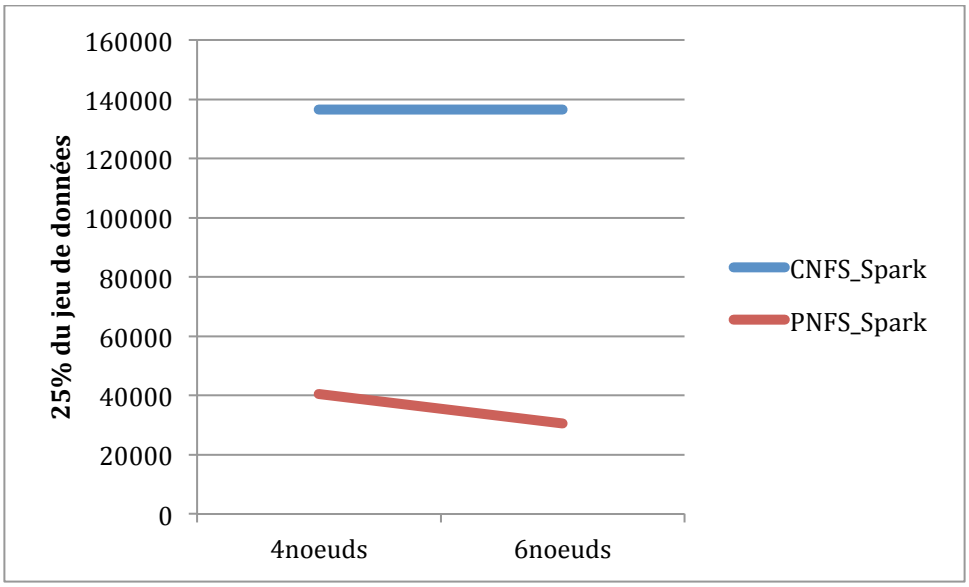

Figure IV.16. - Évolutivité de **PNFS_Spark** et de **CNFS_Spark** avec 25%.

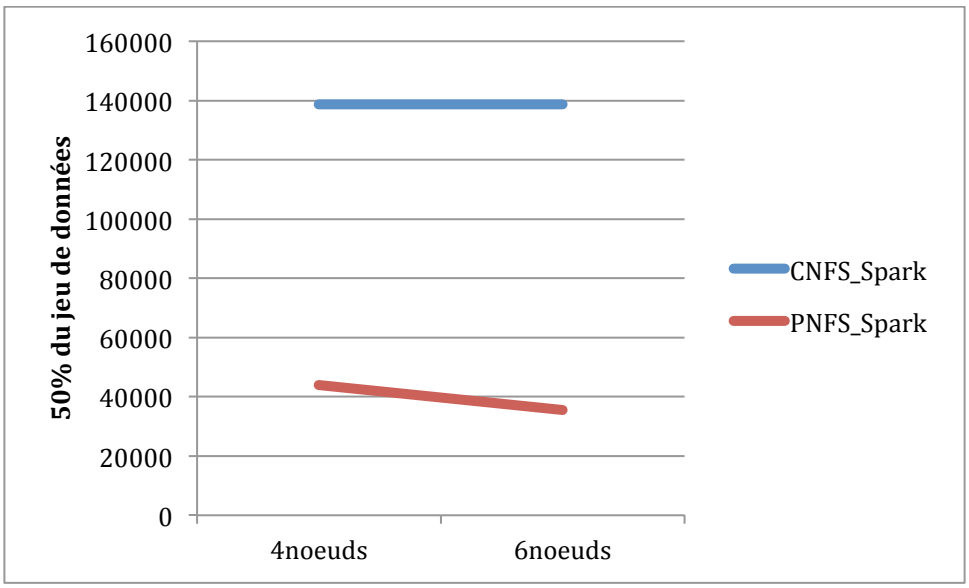

Figure IV.17 - Évolutivité de **PNFS_Spark** et de **CNFS_Spark** avec 50%.

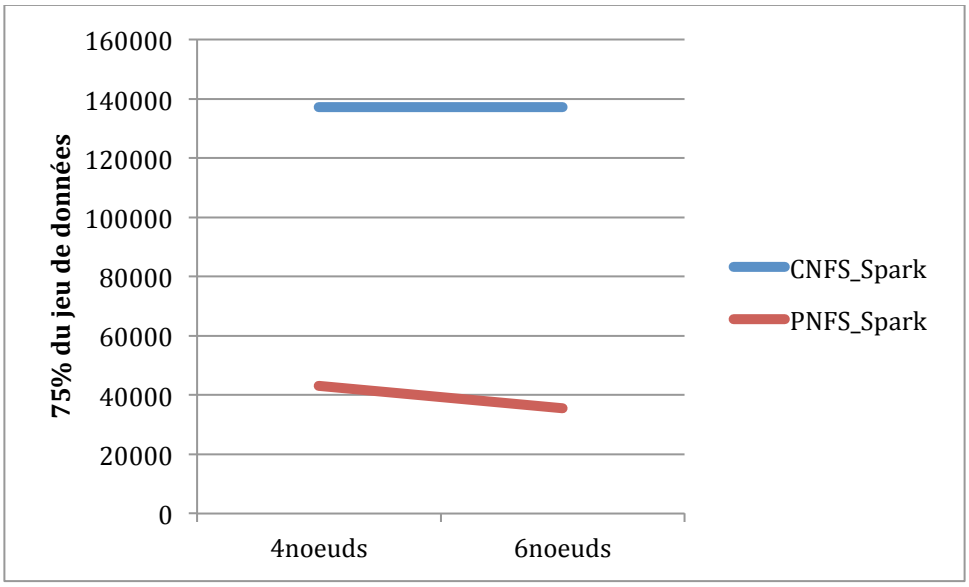

Figure IV.18 - Évolutivité de **PNFS_Spark** et de **CNFS_Spark** avec 75%.

Les figures concernées montrent clairement que le temps d'exécution de PNFS_Spark diminue remarquablement avec l'augmentation du nombre de nœuds alors que le temps pris par CNFS_Spark reste constant.

Les figures qui vont suivre montrent comment le temps varie pour PNFS_Spark et CNFS_Spark, en sélectionnant d'abord 25% puis 50% et après 75% des attributs de colon-tumor.

- ✓ **Colon-Tumor**

Pour colon-tumor, les résultats obtenus avec 4 nœuds et 6 nœuds sont respectivement donnés dans les figures suivantes :

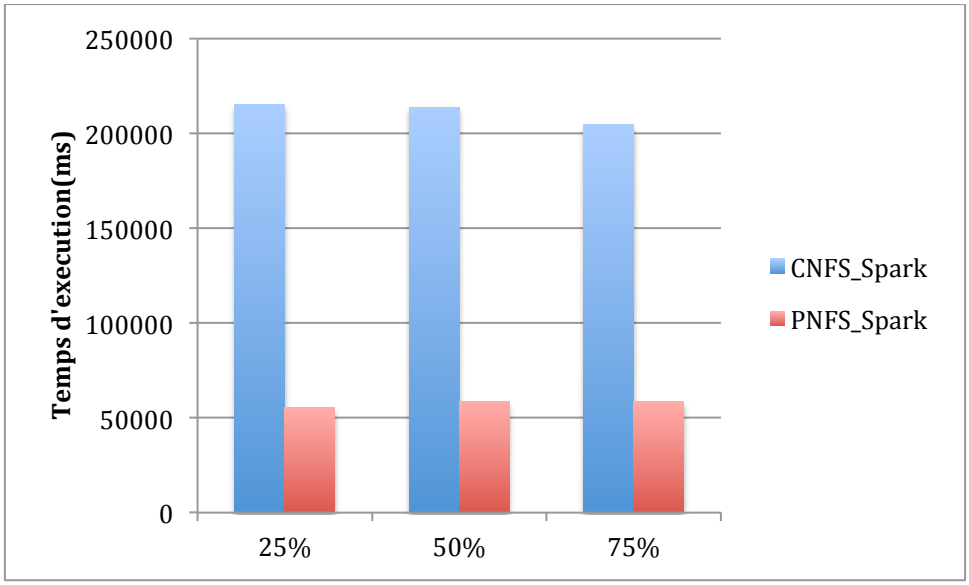

Figure IV.19 - Temps d'exécution de PNFS_Spark pour colon-tumor avec 4 noeuds.

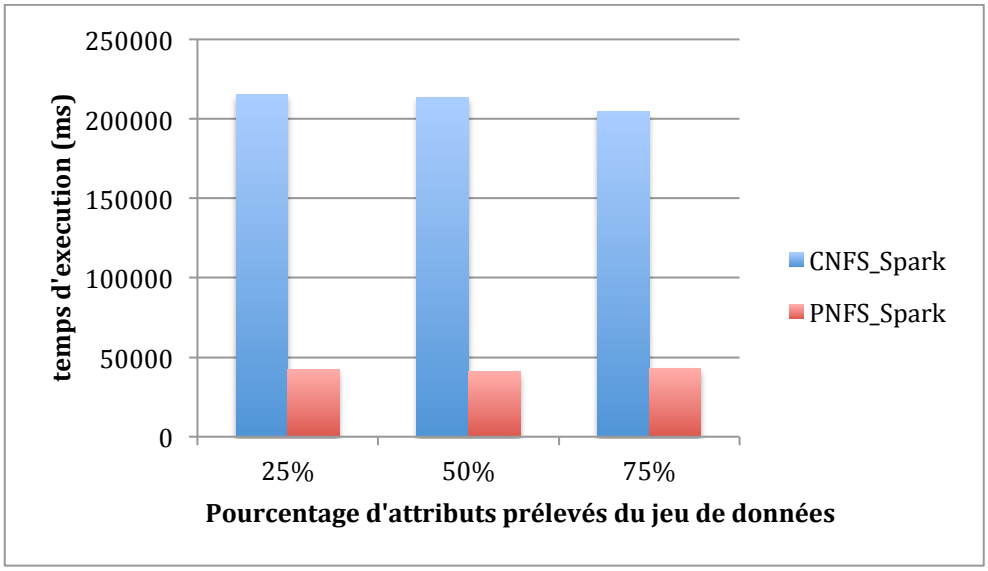

Figure IV.20 - Temps d'exécution de PNFS_Spark pour colon-tumor avec 6noeuds.

Ici aussi nous avons mesuré la métrique d'accélération (speed-up) pour savoir de combien est ce que l'algorithme PNFS_Spark est plus rapide que son équivalent centralisé CNFS_Spark.

Nous mesurons la métrique d'accélération avec la formule suivante :

$$S = \frac{T_{PNFS\_Spark}}{T_{CNFS\_Spark}} \text{ [7]}$$

Où S est le speed-up résultant, $T_{\text{NFS\_Spark}}$ est le temps d'exécution de PNFS_Spark et $T_{C\text{NFS\_Spark}}$ est le temps d'exécution de CNFS_Spark. La valeur trouvée varie entre 4 et 5.

Avec 4 nœuds comme avec 6 nœuds, le temps d'exécution de **PNFS_Spark** est au moins 4 fois plus petit.

3.2.3 PS-RFE-SVM

Dans cette partie, nous allons aussi discuter de l'évolutivité de PS-RFE-SVM, puis nous comparerons son temps d'exécution avec celui de RFE -SVM centralisé.

Les figures **IV.21**, **IV.22** et **IV.23** montrent respectivement comment le temps d'exécution varie en fonction du nombre de nœuds lorsque nous sélectionnons 25%, 50% ou 75% de l'ensemble de données.

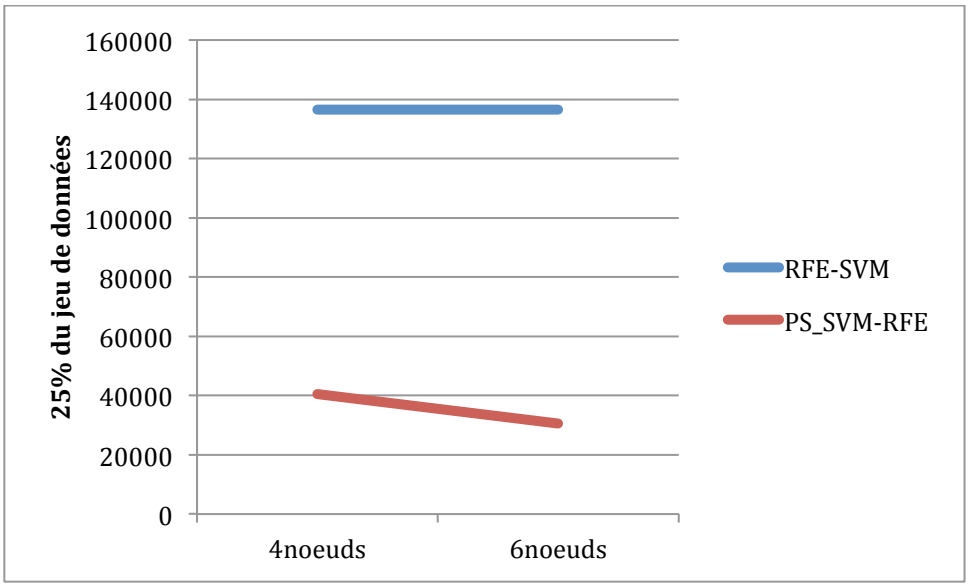

Figure IV.21 - L'évolutivité de PS_SVM-RFE comparé à RFE-SVM classique avec 25%.

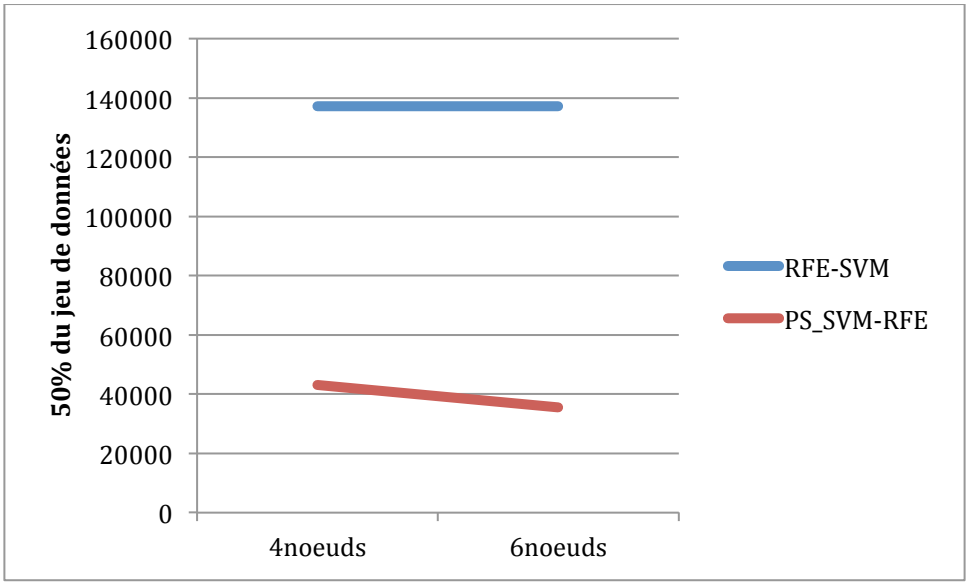

Figure IV.22 - L'évolutivité de PS_SVM-RFE comparé à RFE-SVM classique avec 50%.

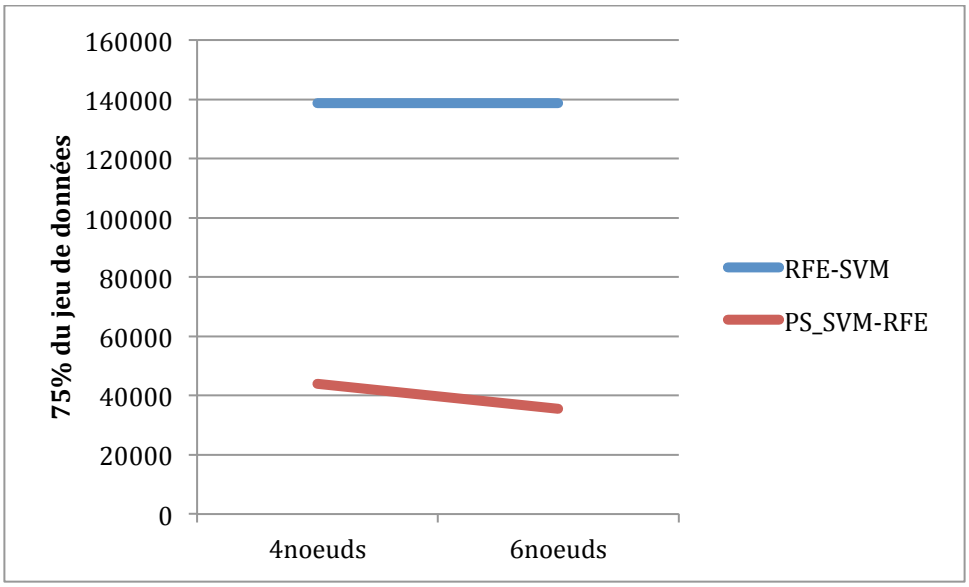

Figure IV.23 - L'évolutivité de PS_SVM-RFE comparé à RFE-SVM classique avec 75%.

Comme avec les méthodes précédentes ici aussi, le temps d'exécution de PS_SVM_RFE diminue considérablement lorsque le nombre de nœuds augmente alors que le temps pris par la méthode RFE-SVM classique reste constant.

Les résultats qui suivent montrent comment le temps varie pour les méthodes PS_SVM_RFE et RFE-SVM, en sélectionnant d'abord 25%, puis 50% et après 75% des caractéristiques pour colon-cancer et colon-tumor.

✓ **Colon-cancer**

Les 2 figures qui suivent indiquent le temps pris par notre méthode comparativement à celui de RFE- SVM centralisé pour respectivement 4 et 6 nœuds.

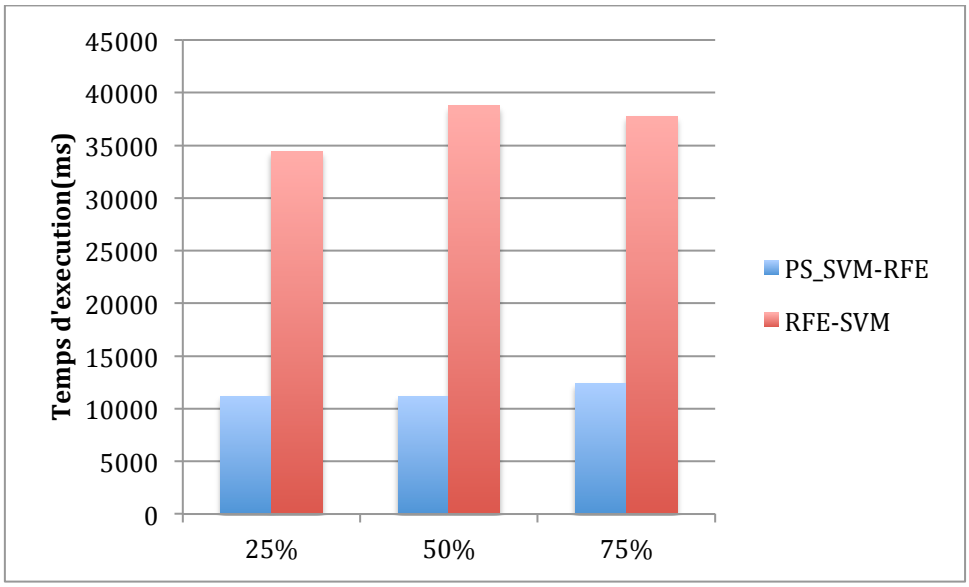

Figure IV.24 - Temps pris par colon-cancer pour 4 nœuds.

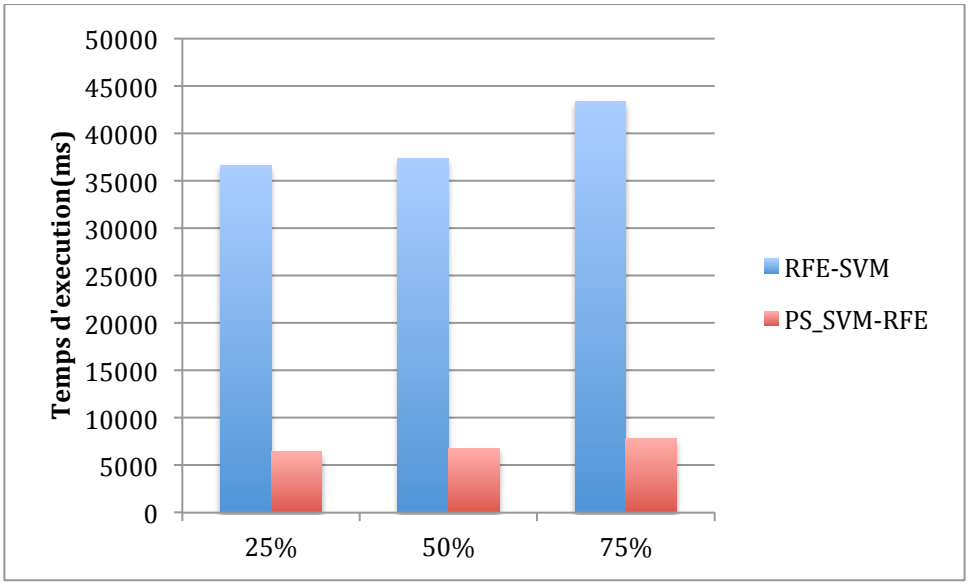

Figure IV.25 - Temps pris par colon-cancer pour 6 nœuds.

Ici aussi, nous avons mesuré la métrique d'accélération (speed-up) pour savoir de combien est ce que l'algorithme PS_SVM_RFE est plus rapide que son équivalent centralisé RFE-SVM.

Nous mesurons la métrique d'accélération avec la formule suivante :

$$S = \frac{T_{\text{PS\_SVM\_RFE}}}{T_{\text{RFE-SVM}}} \quad [7]$$

Où S est le speed-up résultant, $T_{\text{PS\_SVM\_RFE}}$ est le temps d'exécution de PS_SVM_RFE et $T_{\text{RFE-SVM}}$ est le temps d'exécution de RFE-SVM. La valeur de S est entre 4 et 5 en moyenne.

Comme pour PNFS_Spark et PSF-mRMR, on peut constater que, le temps d'exécution de PS_SVM_RFE est au moins 4 fois plus petit que celui de RFE-SVM classique.

- ✓ **Colon-Tumor**

Pour colon-tumor, les résultats obtenus avec 4 nœuds sont donnés à la figure **IV.26**:

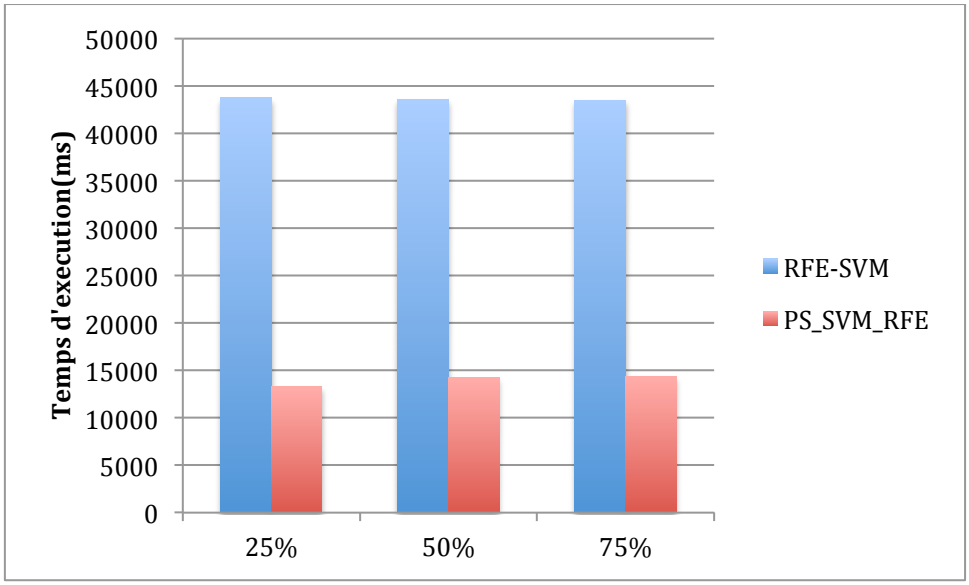

Figure IV.26 - Temps pris par colon-tumor pour 4 nœuds.

Avec un cluster de 6 nœuds, le temps d'exécution est indiqué sur la figure suivante.

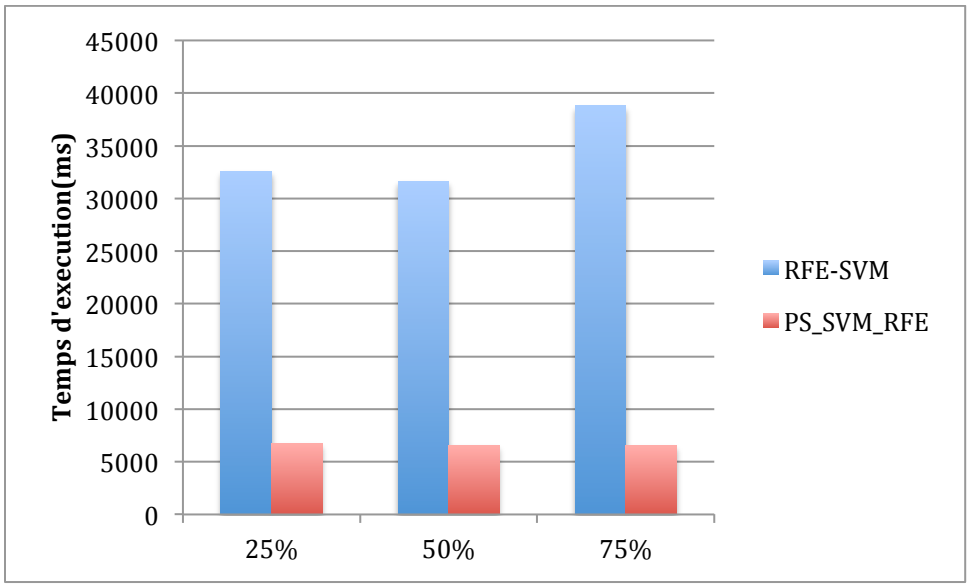

Figure IV.27 - Temps pris par colon-tumor pour 6 nœuds.

Ici aussi, nous avons mesuré la métrique d'accélération (speed-up) pour savoir de combien est ce que l'algorithme PS_SVM_RFE est plus rapide que son équivalent centralisé RFE-SVM.

Nous mesurons la métrique d'accélération avec la formule suivante :

$$S = \frac{T_{\text{PS\_SVM\_RFE}}}{T_{\text{RFE-SVM}}} \quad [7]$$

Où S est le speed-up résultant, $T_{\text{PS\_SVM\_RFE}}$ est le temps d'exécution de PS_SVM_RFE et $T_{\text{RFE-SVM}}$ est le temps d'exécution de RFE-SVM. La valeur de S varie entre 4 et 5.

Comme avec 4 nœuds, le temps d'exécution de PS_SVM-RFE est également au moins 4 fois plus petit.

Par conséquent, nous pouvons conclure de ces expériences que notre solution PS_SVM-RFE surpasse la méthode RFE-SVM centralisée en termes de temps d'exécution. De plus, plus on augmente le nombre de nœuds, plus le temps d'exécution est réduit dans notre méthode alors que celui de la méthode RFE-SVM classique reste constant.

3.2.4 Comparaison des précisions entre nos méthodes et ceux de l'état de l'art

Le tableau **IV.4** montre les précisions obtenues en moyenne avec la méthode **mRMR classique**, la méthode **Fast-mRMR** de l'etat de l'art de méthodes de sélection d'attributs parallèles et les 2 méthodes que nous avons proposées : **PSF-mRMR** et **PNFS-SPARK**. Les jeux de données utilisés pour les expériences sont : colon-cancer et colon-Tumor.

Algorithmes	**Précision du classifieur**	
	Colon-cancer	**Colon-Tumor**
mRMR (classique)	0,8026	0,7949
Fast-mRMR	0,8221	0,8512
PSF-mRMR	0,8221	0,8667
PNFS-SPARK	0,8537	0,8678

Table IV.4- Précision du classifieur avec les jeux de données de référence.

Le tableau montre que les **2 méthodes** que nous avons proposées en l'occurrence **PSF-mRMR** et **PNFS-SPARK** présentent de meilleures précisions par rapport à la méthode **mRMR classique** et la méthode **Fast-mRMR**.

4. Conclusion

Dans ce chapitre, nous avons présenté les différentes méthodes que nous avons proposées pour la sélection d'attributs parallèle. Contrairement à plusieurs méthodes présentées dans le chapitre précédent, les techniques que nous proposons tiennent compte à la fois du temps d'exécution, de la précision et de la scalabilité. Nos résultats ont montré un passage à l'échelle et une bonne performance en terme de temps d'exécution de nos méthodes.

Chapitre V

Proposition d'un algorithme de sélection d'instances parallèle

Dans ce chapitre, nous présenterons la méthode parallèle de sélection d'instances que nous proposons en commençant d'abord par motiver notre proposition.

Tous les détails concernant notre algorithme sont donnés dans la section 2. Dans la section 3, nous évaluons notre proposition en effectuant de nombreuses expériences. Les résultats expérimentaux démontrent que notre méthode est significativement plus efficace et évolutive que d'autres approches dans la littérature.

1. Motivation et aperçu de la proposition

De nos jours, nous sommes entourés par de grandes masses de données qui évoluent à une très grande vitesse et il devient crucial de réduire leur volume pour plusieurs problèmes du monde réel comme l'apprentissage automatique. C'est pourquoi la sélection d'instances joue un rôle essentiel dans le processus d'apprentissage automatique car elle vise à réduire le volume des données en supprimant les instances non pertinentes et redondantes des données d'origine [34]. Les techniques de sélection d'instances permettent de sélectionner des données Big data, un sous-ensemble de données contenant la même ou presque quantité d'informations que sur l'ensemble de données [34]. Cette réduction de données permet ainsi aux algorithmes d'apprentissage automatique classique de traiter les données Big data, après cette étape de prétraitement [34].

C'est ainsi que plusieurs méthodes parallèles, évolutives et efficaces de sélection d'instances ont été proposées dans la littérature. Elles présentent quelques inconvénients comme le fait d'être seulement utilisés avec le classifieur des k-plus-proches-voisins [34]. C'est pour cette raison que nous proposons un algorithme parallèle et évolutif de sélection d'instances que nous avons appelé LSIS (Large Scale

Instance Selection Algorithm for Big Data) et qui est basé sur Spark. LSIS peut s'utiliser avec n'importe quel classifieur.

2. LSIS : notre méthode parallèle de sélection d'instance

Ici, nous expliquons le fonctionnement de la méthode LSIS et les détails de son algorithme.

Le classifieur SVM (Support Vector Machine) est l'un des premiers choix dans l'exploration de données et a été utilisé avec succès pour classer des données séparables linéairement et non linéairement avec une grande précision. Dans la littérature, de nombreuses études révèlent que les caractéristiques de classement utilisant des modèles SVM donnent de bonnes performances. Ainsi, pour obtenir de bonnes performances, dans l'algorithme LSIS, nous combinons le classifieur SVM avec notre critère pour évaluer les instances.

L'algorithme LSIS peut être divisé en trois étapes :

⇒ **<u>À l'étape 1:</u>** Pour chaque instance I_j du jeu de données, chaque attribut a_i rencontré dans l'instance sera associé à la valeur **1**. La valeur **1** signifie que l'attribut a été rencontré dans l'instance.

Cela correspond aux instructions suivantes :

Foreach instance line $I_j \subset I$

Foreach a_i ***in instance line***

mapToPair $(a_i => (a_i, 1))$

EndForeach

EndForeach

Ensuite, pour chaque attribut a_i, la somme des valeurs **1** obtenues est calculée afin d'obtenir le nombre total d'apparition de a_i dans l'ensemble de données, c'est à dire sa fréquence f_i. Cela représente une opération de réduction sous Spark qui se fait comme suit :

$rdd\ [(a_i, f_i)] = reduceByKey(\_ + \_)$

C'est également utilisé pour calculer la fréquence maximale f_{max} des attributs comme suit:

$rdd[(a_i, f_{max})] = rdd[(a_i, f_i)]$.

reduceByKey(math.max(_, _))

⇒ **<u>À l'étape 2:</u>** Le vecteur de poids svm $\boldsymbol{\omega}$ des attributs de l'ensemble de données est d'abord déterminé. Ensuite la moyenne $\boldsymbol{norm(\omega)}$ des valeurs du vecteur de poids $\boldsymbol{\omega}$ est calculée. A l' étape 1, pour chaque instance $\boldsymbol{I_j}$ on avait calculé la fréquence $\boldsymbol{f_i}$ de chaque attribut $\boldsymbol{a_i}$ qui la compose. La fréquence maximale $\boldsymbol{f_{max}}$ des attributs du jeu de données a aussi été déterminée. On peut alors calculer le score de chaque attribut $\boldsymbol{a_i}$ de l'instance $\boldsymbol{I_j}$ avec la formule suivante: $(\boldsymbol{\omega_i/norm(\omega)})/(\boldsymbol{f_{max}-f_i})$**.** Le score de chaque instance I_j est considéré comme la somme des scores de chaque attribut qui la compose. Ainsi, si **n** est le nombre d'attributs alors le score de l'instance $\boldsymbol{I_j}$ s'obtient de la manière suivante :

$$\boldsymbol{Score\ (I_j)=\sum_{i=1}^{n}(\omega_i/norm(\omega))/(f_{max}-f_i)}$$

On applique cette formule pour déterminer le score de chaque instance $\boldsymbol{I_j}$ de l'ensemble de données.

Puis chaque instance $\boldsymbol{I_j}$ est mappée à la paire **($\boldsymbol{I_j}$, score)** :

mapToPair(I_j =>(I_j,score))

⇒ **<u>À l'étape 3:</u>** Pour finir, les workers envoient les paires ***($\boldsymbol{I_j}$, score)*** au maître qui ordonne les scores et retourne les **k** instances avec les meilleurs scores.

Ci-dessous, nous présentons l'algorithme de LSIS.

<u>Algorithm LSIS</u>

Input: Dataset *D* (with *n* instances and *m* attributes)

Output: *k* best inliers

Begin

/*First step*/

Map begin

1:*I=flatmap(line=>f.getInstance())*

2:***Foreach*** *instancelineI$_i$⊂I*

3: *Foreach a$_i$in instance line*

4: *mapToPair (a$_i$=>(a$_i$,1))*

5: *EndForeach*

6:***EndForeach***

End.

Reduce begin

rdd [(a_i, f)] = reduceByKey(_+_)

rdd[(a_i,f_{max})]=rdd[(a_i, f_i)].reduceByKey(math.max(_, _))

End

*/*Second step*/*

Map begin

Foreach *instance line $I_i \subset I$*

mapToPair(I_i=>(I_i,score)) where

$$score = \sum_{i=1}^{m} (w_i / norm\ (w))/(f_{max}-f_i)$$

EndForeach

End

*Return select **instances** according to the **k** highest scores*

End.

3. Expériences

Les expériences ont été réalisées sur un cluster composé de 4 nœuds, chaque nœud ayant 8 cœurs fonctionnant à 2,60 GHz, avec 56 Go de mémoire et un disque de 382 Go, puis sur un cluster de 6 nœuds avec les mêmes paramètres. Les nœuds de calcul s'exécutent tous sur le linux.

3.1 Description des données

Nous avons utilisé le classifieur SVM et LibSVM comme outil SVM.

Nous avons utilisé deux ensembles de données réels de référence choisis sur mldata.org. Certaines informations de ces ensembles de données sont présentées dans le tableau **V.1**.

Nom	Nombre d'attributs	Nombre d'instances
real-sim	72,309	20,958
kddb-raw-libsvm	19, 264,097	1, 163,024

Table V.1- Caractéristiques des ensembles de données de référence.

3.2 Résultats

Pour l'évaluation de la précision de la classification de notre algorithme LSIS, le tableau **V.2** et la figure **V.1** illustrent les résultats obtenus en effectuant différentes réductions du jeu de données : 100%, 70%, 60% et 50% des instances de l'ensemble d'origine.

% réduction	Précision du classifieur	
	kdd-libsvm	real-sim
100	0,8606	0,8923
70	0,8620	0,9243
60	0,8638	0,9263
50	0,8644	0,9260

Table V.2 - Précision du classifieur des jeux de données de référence

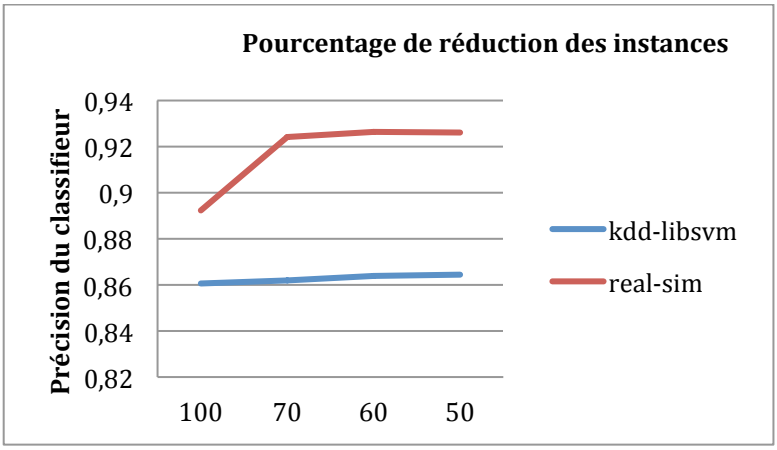

Figure V.1 - Précision du classifieur pour les jeux de données.

Les résultats montrent clairement que la précision est améliorée après la réduction des instances. On constate que les valeurs de précision obtenues par notre méthode dépassent ceux des méthodes de sélection d'instances parallèles présentés dans l'état de l'art au chapitre III section 2. Les meilleurs résultats ont été obtenus avec une sélection de 50% des jeux de données. Ce qui montre que LSIS aide à réduire le volume tout en augmentant la précision du classifieur.

Après avoir discuté de la performance de notre proposition pour la précision de la classification, nous étudions son évolutivité.

Pour ce faire, nous avons varié le nombre de cœurs, et effectué tous les tests avec les mêmes conditions.

La figure suivante montre le temps pris par notre méthode LSIS en fonction du nombre de nœuds utilisés. Les nœuds varient de 2 à 6.

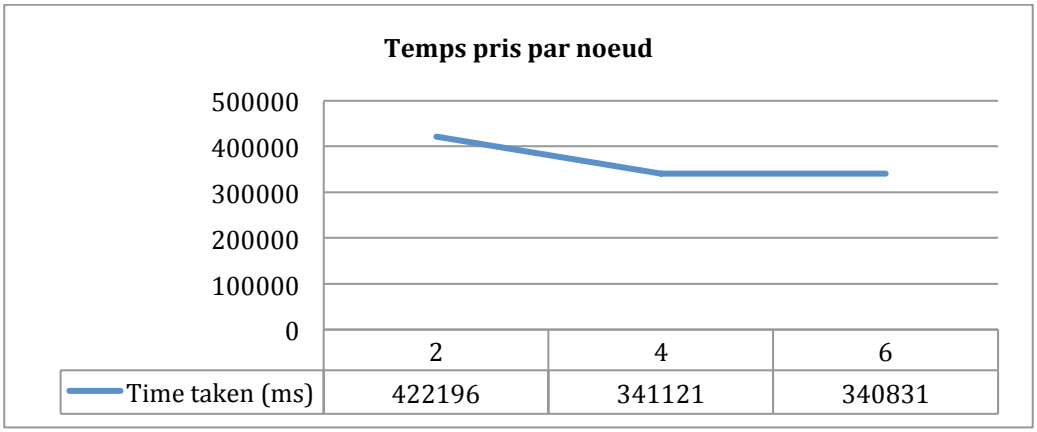

Figure V.2 - évaluation de la scalabilité.

Les résultats montrent que notre solution offre une bonne efficacité de calcul. Le temps de sélection d'instances diminue de manière significative lorsque le nombre de nœuds augmente. Ce qui prouve que notre méthode LSIS passe à l'échelle.

4. Conclusion

Dans ce chapitre nous avons présenté la méthode parallèle de sélection d'instances que nous avons proposé pour faire face à la problématique des BIG DATA. Contrairement aux méthodes présentées au chapitre III, notre algorithme ne se limite pas qu'au classifieur **kNN**. En effet notre méthode n'est pas liée à un classifieur en

particulier. De plus, elle tient compte à la fois du temps d'exécution, de la précision et de la scalabilité. Notre méthode réduit les jeux de données tout en présentant de très bonnes précisions pour le classifieur. En plus, les résultats ont montré la scalabilité de notre proposition et les performances en terme de temps d'exécution.

Conclusion et perspectives

Dans cette partie, nous résumons et discutons les principales contributions apportées dans le cadre de cette thèse. Ensuite, nous donnons des orientations de recherche pour les travaux futurs.

Comme nous l'avons dit dans l'introduction, le problème majeur de nos jours est que la taille des ensembles de données ne cesse d'augmenter tant verticalement et qu'horizontalement. Ainsi il devient crucial de réduire la dimensionnalité et le volume des données pour les problèmes d'apprentissage automatique. C'est pourquoi la sélection d'attributs et la sélection d'instances jouent un rôle essentiel dans le processus d'apprentissage automatique car elles permettent de réduire la haute dimensionnalité et le volume en supprimant les caractéristiques ou instances non pertinentes et redondantes des données d'origine. Cependant, les algorithmes classiques ne sont pas adaptés au Big data car la plupart d'entre eux fonctionnent toujours de manière séquentielle et ne s'adaptent pas bien aux grandes quantités de données. Leur efficacité peut considérablement se détériorer au point de devenir inapplicable. Cela constitue un défi pour ces algorithmes classiques, car il existe un besoin croissant de méthodes évolutives et efficaces.

Ainsi dans cette thèse, nous nous plaçons dans le contexte de la sélection parallèle d'attributs et de la sélection d'instances dans des environnements massivement distribués. L'objectif visé est d'améliorer la précision de la classification, d'accélérer le temps d'exécution et d'obtenir des méthodes de sélection d'attributs et d'instances évolutives lorsque l'on traite de grands ensembles de données.

A cet effet nous avons apporté plusieurs contributions :

1. **PSF-mRMR** : Une version parallèle de mRMR dans laquelle nous avons proposé une métrique qui combine le critère mRMR avec le poids SVM de chaque attribut dans le but d'accroitre la précision de la classification. Le choix de SVM a été motivé par ses performances qui ont été relatées par beaucoup d'auteurs dans la littérature. L'évolutivité et le temps d'exécution de notre méthode ont également été testés et comparés à ceux de la méthode mRMR classique.

Nous avons trouvé dans la littérature, quelques versions parallèles de mRMR, qui visent à réduire le goulot d'étranglement dans la version originale de mRMR qui est lié au calcul de l'information mutuelle entre deux éléments : soit une caractéristique donnée avec la classe, soit une paire d'attributs en entrée. En effet tous les attributs du jeu de données doivent être associés 2 à 2 afin de mesurer leur corrélation et choisir les attributs les moins corrélés aux autres et les plus relevants. Cependant ces méthodes utilisent une approche gloutonne afin de limiter le nombre de comparaisons entre attributs. Ainsi au départ on a un sous ensemble d'attributs à retourner qui est vide. Puis à chaque itération l'attribut considéré comme le meilleur de l'ensemble par la méthode mRMR est ajouté à l'ensemble des attributs à retourner. Le processus s'arrête une fois que le nombre d'attributs à retourner est obtenu. Ainsi l'algorithme itère autant de fois qu'il y a d'attributs à retourner.

De plus ces algorithmes effectuent plusieurs jobs or dans un environnement massivement distribué, la performance globale d'un processus est améliorée quand on peut minimiser le nombre de « jobs » (les « aller/retours » entre les machines distribuées). Cela impacte le temps d'exécution, mais aussi le transfert de données. Dans le cas de la réduction de données, trouver les meilleurs attributs en un seul job simplifié serait donc préférable. C'est pour cette raison que nous avons proposé dans cette thèse une méthode qui en une itération choisit les meilleurs attributs du jeu de données d'entrée et les retourne.

Nos expériences ont montré une meilleure précision par rapport à la méthode mRMR classique et à ses versions parallèles proposées dans la littérature, de même qu'une amélioration du temps d'exécution. De plus notre algorithme a montré une bonne performance en terme de temps d'exécution et de scalabilité.

2. **PS_SVM-RFE** : Une version distribuée et évolutive d'une méthode de sélection de caractéristiques bien connue appelée RFE-SVM. Dans cette méthode, nous donnons un score aux caractéristiques en utilisant

leurs poids SVM puis nous éliminons les caractéristiques avec le plus petit poids dans chaque itération.

Les résultats expérimentaux ont démontré l'évolutivité de PS_SVM-RFE et ses performances en terme de réduction du temps d'exécution en comparaison avec la méthode RFE-SVM classique.

3. **PNFS_Spark** : dans lequel nous proposons une toute nouvelle métrique basée sur l'utilisation de la valeur médiane de la pertinence des attributs avec l'étiquette de classe. Dans cette méthode la relevance de chaque attribut est d'abord calculé et la valeur médiane est déterminée. Les voisins de chaque attribut sont déterminés en se basant sur l'information mutuelle entre les attributs. On considère un attribut comme voisin d'un autre si l'information mutuelle entre les deux est supérieure à la valeur médiane. Le score d'un attribut est alors sa relevance multipliée par son nombre de voisins. Les attributs avec le meilleur score sont alors choisis par notre algorithme.

4. **LSIS** : le principal problème des méthodes de sélection d'instances parallèles est que la plupart d'entre elles sont conçues pour le classifieur des k-plus-proches-voisins, ainsi les instances sélectionnées par ces algorithmes ne sont souvent adaptées qu'à ce type de classifieur.

   LSIS est un tout nouvel algorithme indépendant du classifieur des k-plus-proches-voisins qui peut être appliqué à tout type de classifieur.

   Il a présenté de très bonnes performances en terme de précision de classification et a montré de bons résultats tant en termes de temps d'exécution que de scalabilité comparée aux algorithmes de sélection d'instance de la littérature.

Dans le futur, nous comptons appliquer la parallélisation à plusieurs méthodes classiques de la littérature, qui ont été très peu abordées. C'est le cas par exemple des méthodes « wrapper ». Malgré le fait que ces méthodes présentent de très bonnes performances en comparaison avec celles de type« filtre », elles sont très coûteuses en

terme de temps d'exécution. Cela est dû au fait qu'elles évaluent chaque sous-ensemble généré avec le classifieur. Bien que des propositions aient été faites dans la littérature afin d'améliorer leur performance, la plupart d'entre elles fonctionnent toujours de manière séquentielle.

Dans les méthodes « filtres » également, bien que plusieurs propositions ont été faites, il est possible encore de les améliorer en optimisant certaines de leurs étapes de manière à ce qu'elles puissent fonctionner avec des données encore plus larges.

Nous comptons également soumettre à une conférence, les résultats de la comparaison (en terme de précision de classification et de temps d'exécution) entre LSIS et les méthodes de l'état de l'art de la sélection d'instances parallèle.

Références bibliographiques

[1] Sergio Ramırez-Gallego, Hector Mourino-Talın, David Martınez- Rego,Veronica Bolon-Canedo, Jose Manuel Benıtez, Amparo Alonso- Betanzos and Francisco Herrera. An Information Theory-Based Feature Selection Framework for Big Data under Apache Spark. Journal of latex class files, vol. 13, no. 9, september 2014.

[2] Dilpreet Singh and Chandan K Reddy. A survey on platforms for big data analytics. J Big Data. 2015; 2(1): 8.Published online 2014 Oct 9.

[3] Kyong-Ha Lee, Bongki Moon, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung.Parallel Data Processing with MapReduce: A Survey.SIGMOD Record, December 2011 (Vol. 40, No. 4).

[4] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in OSDI 2004, 2004, pp. 137–150.

[5] Sievert O, Casanova H: A simple MPI process swapping architecture for iterative applications. Int J High Perform Comput Appl 2004,18(3):341–352. 10.1177/1094342004047430.

[6] Dean J, Ghemawat S: MapReduce: simplified data processing on large clusters. Commun ACM 2008,51(1):107–113. 10.1145/1327452.1327492.

[7] Gropp W, Lusk E, Skjellum A (1999) Using MPI: Portable Parallel Programming with the Message-Passing Interface, Vol 1. MIT press.

[8] Lee K-H, Lee Y-J, Choi H, Chung YD, Moon B: Parallel data processing with MapReduce: a survey. ACM SIGMOD Record 2012,40(4):11–20. 10.1145/2094114.2094118.

[9] Thusoo A, Sarma JS, Jain N, Shao Z, Chakka P, Anthony S, Liu H, Wyckoff P, Murthy R: Hive: a warehousing solution over a map-reduce framework. Proceedings of the VLDB Endowment 2009,2(2):1626–1629. 10.14778/1687553.1687609.

[10] K. Sun, W. Miao, X. Zhang, and R. Rao, "An improvement to feature selection of random forests on spark," in IEEE 17th International Conference on Computational Science and Engineering (CSE), 2014, pp. 774–779.

[11] A. Spark, “Apache Spark: Lightning-fast cluster computing,” 2017, [Online; accessed February 2017]. [Online]. Available: https://spark.apache.org/.

[12] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, Learning Spark: Lightning-Fast Big Data Analytics. O’Reilly Media, Incorporated, 2015.

[13] S. del R´ıo, V. Lo´pez, J. M. Ben´ıtez, and F. Herrera, “On the use of mapreduce for imbalanced big data using random forest.” Information Sciences, vol. 285, no. 285, pp. 112–137, 2014.

[14] J. Lin, “Mapreduce is good enough? if all you have is a hammer, throw away everything that’s not a nail!” CoRR, vol. abs/1209.2191, 2012.

[15] A. Fernandez, S. del Rıo, V. Lopez, A. Bawakid, M. J. del Jesus, J. M. Benıtez, and F. Herrera, “Big data with cloud computing: an insight on thecomputing environment, mapreduce, and programming frameworks,” Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, vol. 4,no. 5, pp. 380–409, 2014.

[16] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia. Learning Spark: Lightning-Fast Big Data Analytics. O’Reilly Media, Inc., 1st edition, 2015.

[17] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Technical Report UCB/EECS-2011-82, EECS Department, University of California, Berkeley, July 2011.

[18] Kyong-Ha Lee, Bongki Moon, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung.Parallel Data Processing with MapReduce: A Survey.SIGMOD Record, December 2011 (Vol. 40, No. 4).

[19] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault- tolerant abstraction for in-memory cluster computing. In Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, NSDI’12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.

[20] J. L. Reyes-Ortiz, L. Oneto, and D. Anguita. Big data analytics in the cloud: Spark on hadoop vs mpi/openmp on beowulf. Procedia Computer Science, 53:121 – 130, 2015.

[21] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica. Shark: Sql and rich analytics at scale. In Proc. of 2013 ACM SIGMOD International Conf. on Management of Data, SIGMOD ’13, pages 13–24. ACM, 2013.

[22] Zhao Z., Cox J., Duling D., Sarle W. (2012) Massively Parallel Feature Selection: An Approach Based on Variance Preservation. In: Flach P.A., De Bie T., Cristianini N.

(eds) Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2012. Lecture Notes in Computer Science, vol 7523. Springer, Berlin, Heidelberg.

[23] M. A. Esseghir, G. Goncalves, and Y. Slimani, "Memetic feature selection: Benchmarking hybridization schemata," in HAIS(1), 2010, pp. 351–358

[24] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. J. Mach. Learn. Res. 3, 1157–1182 March (2003).

[25] Guyon, Weston, Barnhill, and Vapnik, "Gene selection for cancer classification using support vector machines," MACHLEARN: Machine Learning, vol. 46, 2002

[26] Malik Yousef, Segun Jung, Louise C. Showe, and Michael K. Showe. Recursive cluster elimination (RCE) for classification and feature selection from gene expression data. BMC Bioinformatics 8 (2007).

[27] Carlos Cotta and Pablo Moscato. The -feature set problem is [2] complete. J. Comput. Syst. Sci. 67(4), 686–690 (2003).

[28] J. C. H. Hernandez, B. Duval, and J.-K. Hao, "Svm based local search for gene selection and classification of microarray data," in BIRD, 2008, pp. 499- 508.

[29] Piyushkumar A. Mundra and Jagath C. Rajapakse. SVM-RFE with relevancy and redundancy criteria for gene selection. , 4774, pages 242–252. Springer (2007).

[30] Yuchun Tang, Yan-Qing Zhang, and Zhen Huang. Development of twostage SVM-RFE gene selection strategy for microarray expression data analysis. IEEE/ACM Trans. Comput. Biology Bioinform 4(3), 365–381 (2007).

[31] Fodé Camara, Mouhamadou Lamine Samb, Samba Ndiaye and Yahya Slimani. Privacy Preserving RFE-SVM for Distributed Gene Selection. IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 1, No 2, January 2012, ISSN (Online): 1694-0814, www.IJCSI.org.

[32] Priya Dahiya 1, Chaitra.B 2and Usha Kumari. Survey on Big Data using Apache Hadoop and Spark.International Journal of Computer Engineering In Research Trends, 4(6):pp:195-201,June -2017.

[33] Reine Marie Marone; Fodé Camara; Samba Ndiaye. A large-scale filter method for feature selection based on spark. 2017 IEEE 4th International Conference on Soft Computing & Machine Intelligence (ISCMI).

[34] Reine Marie Marone; Fodé Camara; Samba Ndiaye. LSIS: Large scale instance selection algorithm for big data. 2017 3rd IEEE International Conference on Computer and Communications (ICCC).

[35] Mouhamadou Lamine SAMB, Fodé CAMARA, Samba NDIAYE Yahya SLIMANI, Mohamed Amir ESSEGHIR. Approche de sélection d'attributs pour la classification basée sur l'algorithme RFE-SVM. Special issue CARI'12.

[36] Hassan CHOUAIB. Sujet de thèse: Sélection de caractéristiques: méthodes et applications. Université Paris Descartes UFR de Mathématiques et Informatique ED Informatique, Télécommunications et Electronique, Soutenue le 8 juillet 2011.

[37] Almuallim, H. et Dietterich, T. G. (1991). Learning with many irrelevant features. In In Proceedings of the Ninth National Conference on Artificial Intelligence, pages 547–552. AAAI Press. Almuallim, H. et Dietterich, T. G. (1992).

[38] Efficient algorithms for identifying relevant features. In In Proceedings of the Ninth Canadian Conference on Artificial Intelligence, pages 38–45.

[39] Morgan Kaufmann. Sikora, R. et Piramuthu, S. (2007,). Framework for efficient feature selection in genetic algorithm based data mining. European Journal of Operational Research, 180(2):723–737.

[40] Somol, P., Pudil, P. et Kittler, J. (2004). Fast branch & bound algorithms for optimal feature selection. IEEE Pattern Analysis and Machine Intelligence, 26:900–912.

[41] Liu, H. et Yu, L. (2005). Toward integrating feature selection algorithms for classification and clustering. IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEE- RING, 17:491–502.

[42] Peng, H., Long, F. et Ding, C. (2005). Feature selection based on mutual information : criteria of max-dependency, max-relevance, and min-redundancy. IEEE Transactions on Pattern Analysis and Machine Intelligence, 27:1226–1238.

[43] Ritthoff, O., Klinkenberg, R., Fischer, S. et Mierswa, I. (2002). A hybrid approach to feature selection and generation using an evolutionary algorithm. In In Proc. 2002 U.K. Workshop on Computational Intelligence (UKCI-02, pages 147–154. University of.

[44] Somol, P., Pudil, P. et Kittler, J. (2004). Fast branch & bound algorithms for optimal feature selection. IEEE Pattern Analysis and Machine Intelligence, 26:900–912.

[45] Oliveira, L. S., Sabourin, R., Bortolozzi, F. et Suen, C. Y. (2002). Feature selection using multi-objective genetic algorithms for handwritten digit recognition. In Proceedings of the 16 th International Conference on Pattern Recognition (ICPR'02) Volume 1 - Volume 1, ICPR '02, Washington, DC, USA. IEEE Computer Society.

[46] M.sujatha, Dr. G. Lavanya Devi. Feature Selection Techniques using for High Dimensional Data in Machine Learning. International Journal of Engineering

Research & Technology (IJERT), ISSN: 2278-0181, Vol. 2 Issue 9, September – 2013.

[47] Fodé CAMARA. Proposition et mise en oeuvre d'un protocole de sécurité dans un processus de datamining. Sous la direction de Yahya SLIMANI, professeur à l'université de Tunis. Soutenue en 2012.

[48] Chuan Liu, Wenyong Wang, Qiang Zhao andMartin Konan. A new feature selection method based on a validity index of feature subset.Pattern Recognition Letters,Volume 92, 1 June 2017, Pages 1-8.

[49] Wenyan Z, Xuewen L , Jingjing W. Feature Selection for Cancer Classication Using Microarray Gene Expression Data. Biostat Biometrics Open Acc J. 2017;1(2): 555557.

[50] Fouille de données notes de cours ph. preux université de lille. http://www.grappa.univ-lille3.fr/~ppreux/fouille.

[51] Andronicus A. Akinyelu and Aderemi O. Adewumi.Improved Instance Selection Methods for Support Vector Machine Speed Optimization.Security and Communication Networks,Volume 2017 (2017), Article ID 6790975, 11 pages.

[52] S. Garcia, J. Derrac, J. Cano, F. Herrera. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. Pattern Anal. Mach. Intell. IEEE Trans., 34 (3) (2012), pp. 417–435 http://dx.doi.org/10.1109/TPAMI.2011.142

[53] Carbonera, Joel Luis, and Mara Abel. A novel density-based approach for instance selection. IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI), 2016.

[54] E. Leyva, A. González, and R. Pérez, Three new instance selection methods based on local sets: A comparative study with several approaches from a bi-objective perspective, Pattern Recognition, vol. 48, no. 4, pp. 1523–1537, 2015.

[55] Carbonera, Joel Luis, and Mara Abel. A density-based approach for instance selection. IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI), 2015.

[56] Srisawat, A., Phienthrakul, T., Kijsirikul, B. SV-kNNC: An Algorithm for Improving the Efficiency of k-Nearest Neighbor. PRICAI'06 proceedings of the 9th Pacific Rim international conference on Artificiel Intelligence. Guilin, China — August 07 - 11, 2006.

[57] P. Hart. The condensed nearest neighbor rule (corresp.). Inf. Theor. IEEE Trans., 14 (3) (1968), pp. 515–516.

[58] J. A. Olvera-López, J. A. Carrasco-Ochoa, J. F. Martínez-Trinidad, and J. Kittler, “A review of instance selection methods,” Artificial Intelligence Review, vol. 34, no. 2, pp. 133–143, 2010. View at Publisher · View at Google Scholar · View at Scopus

[59] Y. Li, Z. Hu, Y. Cai, and W. Zhang, “Support vector based prototype selection method for nearest neighbor rules,” in Proceedings of the International Conference on Natural Computation, pp. 528–535, Berlin, Germany, 2005.

[60] J. Chen, C. Zhang, X. Xue, and C.-L. Liu, “Fast instance selection for speeding up support vector machines,” Knowledge-Based Systems, vol. 45, pp. 1–7, 2013.

[61] C.-H. Chou, B.-H. Kuo, and F. Chang, “The generalized condensed nearest neighbor rule as a data reduction method,” in Proceedings of the 18th International Conference on Pattern Recognition (ICPR '06), pp. 556–559, IEEE, Hong Kong, August 2006.

[62] S. García, J. Derrac, I. Triguero, C. J. Carmona, and F. Herrera, “Evolutionary-based selection of generalized instances for imbalanced classification,” Knowledge-Based Systems, vol. 25, no. 1, pp. 3–12, 2012.

[63] J. R. Cano, F. Herrera, and M. Lozano, “Using evolutionary algorithms as instance selection for data reduction in KDD: an experimental study,” IEEE Transactions on Evolutionary Computation, vol. 7, no. 6, pp. 561–575, 2003.

[64] T. Raicharoen and C. Lursinsap, “A divide-and-conquer approach to the pairwise opposite class-nearest neighbor (POC-NN) algorithm,” Pattern Recognition Letters, vol. 26, no. 10, pp. 1554–1567, 2005.

[65] D. R. Wilson and T. R. Martinez, “Reduction techniques for instance-based learning algorithms,” Machine Learning, vol. 38, no. 3, pp. 257–286, 2000.

[66] H. Zhang and G. Sun, “Optimal reference subset selection for nearest neighbor classification by tabu search,” Pattern Recognition, vol. 35, no. 7, pp. 1481–1490, 2002.

[67] Koggalage, R. and Halgamuge, S., Reducing the number of training instances for fast support vector machine classification. Neural Information Processing Letters and Reviews. v2 i3. 57-65.

[68] Chih-Fong Tsai , Kai-Chun Cheng, Simple instance selection for bankruptcy prediction, Knowledge-Based Systems, 27, p.333-342, March, 2012 [doi>10.1016/j.knosys.2011.09.017].

[69] Salvador Garcıa , Joaquın Derrac , Isaac Triguero , Cristóbal J. Carmona , Francisco Herrera, Evolutionary-based selection of generalized instances for imbalanced

classification, Knowledge-Based Systems, v.25 n.1, p.3-12, February, 2012 [doi>10.1016/j.knosys.2011.01.012]

[70] Fabrizio Angiulli , Annabella Astorino, Scaling up support vector machines using nearest neighbor condensation, IEEE Transactions on Neural Networks, v.21 n.2, p.351-357, February 2010 [doi>10.1109/TNN.2009.2039227] .

[71] Anthony Kleerekoper, Michael Pappas, Adam Pocock, Gavin Brown, Mikel Lujan. A scalable implementation of information theoretic feature selection for high dimensional data. Big Data (Big Data), 2015 IEEE International Conference, USA.

[72] Ramírez-Gallego, Sergio and Lastra, I and Martinez, David and Bolón-Canedo, Verónica and Benítez, José and Herrera, Francisco and Alonso-Betanzos, Amparo. Fast-mRMR: Fast Minimum Redundancy Maximum Relevance Algorithm for High-Dimensional Big Data: FAST-mRMR ALGORITHM FOR BIG DATA. International Journal of Intelligent Systems, July 2016, DOI: 10.1002/int.21833.

[73] Daniel Peralta,Sara del Río,Sergio Ramírez-Gallego,Isaac Triguero, Jose M. Benitezand Francisco Herrera.Evolutionary Feature Selection for Big Data Classification: A MapReduce Approach. Mathematical Problems in Engineering Volume 2015 (2015), Article ID 246139, 11 pages.

[74] Osmar R. Zaïane, Mohammad El-Hajj, and Paul Lu. Fast parallel association rule mining without candidacy generation. In Proceedings of IEEE International Conference on Data Mining, pages 665–668, San Jose, California, USA, 2001.

[75] Eric Li and Li Liu. Optimization of frequent itemset mining on multiple-core processor. In Proceedings of International Conference on Very Large Data Bases, pages 1275–1285, Vienna, Austria, 2007.

[76] Piyushkumar A. Mundra and Jagath C. Rajapakse.SVM-RFE With MRMR Filter for Gene Selection.IEEE transactions on nanobioscience, vol. 9, no. 1, march 2010.

[77] Yin-Wen Chang and Chih-Jen Lin. Feature ranking using linear SVM.Proceedings of the Workshop on the Causation and Prediction Challenge at WCCI 2008, PMLR 3:53-64, 2008.

[78] Á. Arnaiz-González, J.-F. Díez-Pastor, J.J. Rodríguez, C. García-Osorio. Instance selection of linear complexity for big data. Knowl. Based Syst., 000 (2016), pp. 1–13.

[79] Álvar Arnaiz-González, Alejandro González-Rogel, José-Francisco Díez-Pastor, Carlos López-Nozal. MR-DIS: democratic instance selection for big data by MapReduce. Artificial Intelligence (2017), pp. 1-9, doi:10.1007/s13748-017-0117-5.

[80] Junhai Zhai, Xizhao Wang, Xiaohe Pang. Voting-based instance selection from large data sets with MapReduce and random weight networks. Information Sciences. Volumes 367-368, Pages 1066-1077.

[81] Isaac Triguero, Daniel Peralta, Jaume Bacardit, Salvador García, Francisco Herrera, MRPR: A MapReduce solution for prototype reduction in big data classification.Neuro Computing, Volume 150, Part A, 20 February2015, Pages 331–345.

[82] Alexandros Labrinidis and H. V. Jagadish. Challenges and opportunities with big data. Proc. VLDB Endow., 5(12):2032–2033, August 2012.

[83] Saber Salah. Sujet de these: Parallel Itemset Mining in Massively Distributed Environments. Thèse pour obtenir le grade de docteur. Délivré par l'Université de Montpellier. Préparée au sein de l'ecole doctorale I2S et de l'unité de recherché UMR 5506. Soutenue le 20/04/2016.

[84] http://mldata.org/repository/data/viewslug/ovarian-cancer-nci-pbsii-data/.