Ministère de l'enseignement Supérieur et de la recherche Scientifique
وزارة التعليم العالي والبحث العلمي

Badji Mokhtar Annaba University
Université Badji Mokhtar – Annaba
Faculté de Technologie

Département d'Informatique

جامعة باجي مختار – عنابـــــــــة
كلية التـكـنـولـوجـيـا
قسـمـــــ الاعلام الآلي

Thèse

Présentée pour obtenir le diplôme de

Doctorat

Spécialité : Ingénierie des Systèmes Complexes (ISC)

Filière : INFORMATIQUE

Par :

Mr HAOUARI Ahmed Taha

Thème :

Approches Bio-inspirées pour la Fouille de Données en Ingénierie de Logiciels

Thèse soutenue le …….…… devant le jury composé de :

N°	Nom et prénom	Grade	Etablissement	Qualité
01	TALEB Nora	Prof.	Université Badji Mokhtar -Annaba	Président
02	SOUICI-MESLATI Labiba	Prof.	Université Badji Mokhtar -Annaba	Rapporteur
03	ATIL Fadila	Prof.	Université Badji Mokhtar -Annaba	Co-rapporteur
04	AMIRAT Abdelkrim	Prof.	Université de Souk Ahras	Examinateur
05	BOUNOUR Nora	Prof.	Université Badji Mokhtar -Annaba	Examinateur

Dédicaces

Ce travail est dédié à ma mère Habachi Amel et mon père Haouari Mohammed, sans vous mes rêves seraient un lointain souvenir, sans votre patience et votre courage, je n'aurai jamais pu aboutir. Vous m'avez appris à toujours rester droit et faire mon travail de la meilleure façon possible. Vous m'avez appris à me relever quand je suis au plus bas, à rire et apprécier chaque moment de ma vie.

À ma Mère, merci de croire en moi, de toujours me pousser à me dépasser, merci pour l'amour que tu m'apportes chaque jour et pour tes bons plats. Notre voyage à Istanbul est le plus beau souvenir de ma vie.

À mon Père merci de m'avoir appris à m'exprimer et m'imposer, merci de m'avoir appris l'amour du travail bien fait, les souvenir des journées à regarder des matches de football avec toi sont les plus heureux de ma vie.

Je sais que je ne suis pas le fils parfait mais j'espère que ce travail vous rendra un peu du bonheur que vous m’avez procuré. Merci pour tout…

Remerciements

Mes sincères remerciements au Professeur Souici-Meslati ainsi qu'au Professeur Atil de m'avoir pris sous vos ailes, d'avoir accepté mes caprices et donné carte blanche pour m'exprimer pleinement, ce fut difficile mais Hamdouli Allah. Au Professeur Meslati Djamel "Rabi yarhamhu", merci d'avoir été un modèle à suivre et de m'avoir donné un but à atteindre, dans ma vie. Je n'aurai jamais pensé être là où je suis maintenant sans votre orientation. Merci au Docteur Benouhiba de m'avoir appris à aimer l'informatique à l'époque de ma licence, je n'étais pas très motivé pour continuer dans ce domaine.

Je remercie vivement les Professeurs Amirat Abdelkrim, Taleb Nora et Bounour Nora pour l'intérêt qu'ils ont bien voulu porter à mon travail en acceptant de faire partie de mon jury de thèse.

Merci à ma grand-mère "Nana Becha", j'étais petit quand je t'ai appelé par ce surnom et jusqu'à ce jour je continue à le faire, mon affection pour toi est presque aussi forte que celle d'un fils à sa mère. Merci à mon meilleur ami Chames Eddin de m'avoir aidé à supporter ces 5 années de stress. Merci à Rassim d'avoir été à mes côtés tout au long de mes années Fac, je me souviens de notre retour à la maison par le train, tristes lorsque nous avions de mauvaises notes dans nos examens. Nous étions 6 à cette époque et nous voilà les 2 derniers du groupe.

Merci au Docteur Kilani de m'avoir donné la chance de faire le plus beau travail au monde, être enseignent est l'expérience la plus enrichissante de ma vie. Merci à toi Asma, de m'avoir appris à quoi ressemble le vrai courage, les choses par lesquelles tu es passée m'ont montré à quel point tu es forte et je suis très heureux de t'appeler mon amie.

Merci à mon cousin Akram d'être le grand frère dont j'ai besoin. Merci à mes petits frères Yacine, Idriss et Anes d'être là pour moi, j'espère avoir été un bon modèle pour vous. Merci à tous mes amis pour m'avoir accompagné depuis que je suis enfant Reda, Said, Monder, Mounir et tant d'autres. Merci à tous les membres de ma famille mon Oncle Mourad, Saddek, Tante Aicha et tant d'autres, sans oublier mon cousin Khalil pour m'avoir laissé travailler dans ton bureau.

Enfin, un énorme Merci à Dieu pour tout ce que j'ai ….

« Approches Bio-inspirées pour la Fouille de Données en Ingénierie de Logiciels »

Résumé :

Le sujet abordé dans cette thèse se situe dans le domaine de l'ingénierie logicielle empirique et, plus précisément, celui de l'utilisation d'approches de fouille de données et d'apprentissage automatique, principalement bio-inspirées pour la prédiction de défauts logiciels.

Les techniques de prédiction de défauts visent à réduire le coût de la maintenance des logiciels, en détectant, le plus tôt possible, les défauts qui risquent d'apparaitre après l'étape de livraison du logiciel. La prédiction des défauts consiste à recourir à un modèle permettant de déterminer la possibilité de présence de défauts dans un logiciel à travers des analyses liées à une base de données intra ou inter applications. Ceci mène à identifier l'ensemble de composants du logiciel susceptibles de contenir un défaut lors de la prochaine livraison du produit.

Suite à notre étude des concepts et méthodes liés à la prédiction de défauts logiciels, en nous intéressant particulièrement aux approches bio-inspirées, nous nous sommes concentrés sur les systèmes immunitaires artificiels qui nous ont semblés particulièrement attrayants au vu de leur succès dans plusieurs voies de recherche. Nous avons constaté que, malgré un nombre non négligeable de travaux, très peu ont étudié la capacité des approches immunologiques à prédire les défauts logiciels.

Ceci nous a incités à conduire plusieurs expérimentations relatives aux différentes catégories de recherches dans ce domaine. En premier lieu, nous avons comparé les performances des algorithmes immunologiques à d'autres méthodes référencées dans un scénario Intra-projets, tout en essayant d'implémenter un outil pour aider les développeurs dans la prédiction de défauts Cross-projets. Par la suite, nous avons mené une très large étude empirique comparant et validant les performances des méthodes immunologiques dans un environnement Inter-projets, en utilisant des tests statistiques pour nous assurer de la validité empirique de nos conclusions.

Les résultats obtenus sont très encourageants, et l'outil proposé arrive à améliorer le taux de classification. Parmi les algorithmes sélectionnés lors de notre évaluation, nous avons constaté que la version hybride de l'algorithme Immunos-81 est très intéressante pour prédire les défauts Inter-projets des logiciels critiques qui sont en cours de développement.

Mots clés : Approches immunologiques, Prédiction de défauts logiciels, Qualité des logiciels, Classification, Apprentissage automatique.

"مناهج مستوحاة من الطبيعة للتنقيب عن البيانات في هندسة البرمجيات "

الملخص:
الموضوع الذي تتم معالجته في هذه الأطروحة ينتمي إلى مجال هندسة البرمجيات التجريبية، وبشكل أكثر تحديدًا، استخدام مناهج التنقيب عن البيانات والتعلم الآلي ، المستوحاة أساسًا من علم الأحياء، للتنبؤ بأخطاء البرامج.
تهدف تقنيات التنبؤ بالعيوب إلى تقليل تكلفة صيانة البرامج من خلال الكشف، في أقرب وقت ممكن، عن العيوب التي من المحتمل أن تظهر بعد مرحلة تسليم البرنامج. يتكون التنبؤ بالأخطاء من استخدام نموذج لتحديد إمكانية وجود أخطاء في البرنامج من خلال التحليلات المرتبطة بقاعدة بيانات داخل أو بين التطبيقات. يؤدي هذا إلى تحديد مجموعة مكونات البرنامج التي من المحتمل أن تحتوي على عيب أثناء التسليم التالي للمنتج.
بعد دراستنا للمفاهيم والأساليب المتعلقة بالتنبؤ بأخطاء البرامج، مع اهتمام خاص بالمناهج المستوحاة من الأحياء، ركزنا على أنظمة المناعة الاصطناعية التي بدت جذابة بشكل خاص بالنسبة لنا نظرًا لنجاحها في العديد من مجالات البحث. وجدنا أنه على الرغم من العدد الكبير من الأعمال ، فإن القليل جدًا منهم قد حقق في قدرة الأساليب المناعية على التنبؤ بعيوب البرامج.
دفعنا هذا إلى إجراء العديد من التجارب المتعلقة بفئات البحث المختلفة في هذا المجال. أولاً ، قمنا بمقارنة أداء الخوارزميات المناعية بالطرق المرجعية الأخرى في سيناريو داخل المشروع ، أثناء محاولة تنفيذ أداة لمساعدة المطورين في التنبؤ بالعيوب بين المشاريع. بعد ذلك ، أجرينا دراسة تجريبية كبيرة جدًا للمقارنة والتحقق من أداء الأساليب المناعية في بيئة مشتركة بين المشاريع ، باستخدام الاختبارات الإحصائية لضمان الصلاحية التجريبية لاستنتاجاتنا.
نتائج تجاربنا كانت مشجعة حيث حسنت الأداة المقترحة معدل التصنيف. من بين جميع الخوارزميات التي اختبرناها، وجدنا أن النسخة المختلطة من خوارزمية Imunnos-81 هي الأنسب للتنبؤ بعيوب ما بين الإصدارات للبرامج الحرجة التي هي في حالة التطوير.

كلمات مفتاحية: المناهج المناعية ، التنبؤ بعيوب البرامج، جودة البرامج، التصنيف، التعلم الالي .

« Bio-inspired Approaches for Data Mining in Software Engineering »

Abstract :

The topic addressed in this thesis is in the field of empirical software engineering and, more specifically, the use of data mining and machine learning approaches, mainly bio-inspired, for the prediction of software defects.

Software fault prediction techniques aim to reduce the cost of software maintenance by detecting, as early as possible, the defects that are likely to appear after the software delivery stage. Fault prediction consists of using a model to determine the possibility of the presence of faults in software through analyzes linked to an intra or inter-application database. This leads to identify the set of software components which could contain a defect during the next product delivery.

Following our study of the concepts and methods related to the software faults prediction, with a particular interest in bio-inspired approaches, we focused on artificial immune systems which seemed particularly attractive to us in view of their success in several research domains. We found that, despite a significant number of works, very few have investigated the ability of immunological approaches to predict software defects.

This encouraged us to conduct several experiments relating to the different categories of research in this field. First, we compared the performance of immunological algorithms to other referenced methods in an intra-project scenario, while trying to implement a tool to help developers in cross-project defect prediction. Subsequently, we conducted a very large empirical study comparing and validating the performance of immunological methods in an inter-project environment, using statistical tests to ensure the empirical validity of our conclusions.

The obtained results are very encouraging, and the proposed tool succeeds in improving the classification rate. Amongst all the selected algorithms during our study, we have found that the hybrid version of the Immunos-81 algorithm is the most suitable for predicting inter-projects defects in critical software under development.

Key words : Immunological approaches, Software fault prediction, Software quality, Classification, Machine learning.

Sommaire

Table des Figures

Liste des Tableaux

Liste des abréviations

AB	Anticorps	Antibody
AG	Antigène	Antigen
AIRS	Système immunitaire artificiel pour la classification	Artificial Immune Recognition System
AIS	Systèmes Immunitaires Artificiels	Artificial Immune Systems
ANN	les réseaux de neurones artificiels	Artificial Neural Network
AUC	l'aire Sous la courbe ROC	Area Under the ROC Curve
CC	complexité cyclomatique	Cyclomatic complexity
CK	Les métriques orientés objets de Chidamber et Kemerer	Chidamber et Kemerer Software Metrics
CLONALG	Algorithmes de la Sélection Clonale	CLONal selection ALGorithm
CSCA	Algorithmes de la Sélection clonale pour la classification	Clonal Selection Classification algorithm
CPDP	Prédiction de défauts logiciels Cross-projets	Cross-project Software Faults Prediction
CV	Prédiction de défauts logiciels intra-projets	Cross Version/Intra-project defect prediction
DM	Exploration de données/ Fouille de données	Data Mining
GA	l'algorithme génétique	Genetic algorithm
GL/SE	Génie logiciel	Software Engineering
IA	Intelligence Artificielle	Artificial intelligence
IBK/KNN	Méthode des k plus proches voisins	k-nearest neighbors algorithm
J48	L'algorithme d'Arbres de décisions	Decision tree Algorithm
LOC	Nombre de lignes de code	number of lines of code
LR	Régression Logistique	Logistic regression
MC	Cellule Mémoire	Memory Cell
MDP	Métriques de processus	Process Metrics
MHAL	Métriques de Halstead	Halstead Software Metrics
ML	Apprentissage automatique	Machine Learning
MLP	Perceptron Multicouche	Multilayer Perceptron
NB	Classification par la méthode du Naïve Bayésienne	Naive Bayes classifier
OO	Orienté Objet	Object-oriented
PD	Mesure de performance Rappel	Recall Performance Metrics

PR	Mesure de performance Précision	Precision Performance Metrics
PV	Prédiction de défauts logiciels Inter-projets	Previous version /Inter- defect prediction
QL/SQ	Qualité logicielle	Software Quality
RF	L'algorithme de Forêt d'arbres décisionnels	Random forest Algorithm
SDLC	Cycle de vie d'un logiciel	Systems development life cycle
SFP	Prédiction de défauts logiciels	Software Fault Prediction
SM	Mesures Logicielles	Software Measurement
SQA	Méthode d'Assurance Qualité Logicielle	Software Quality Assurance
ST	Test logiciel	Software Testing
SVM	L'algorithme de Machine à vecteurs de support	Support-Vector Machine Algorithm
WEKA	Environnement Waikato pour l'analyse de connaissances	Waikato environment for knowledge analysis

Introduction

Au milieu des années 30, les travaux d'Alan Turing ont poussés la science informatique dans une ère nouvelle. Son concept "Machine de Turing Universelle" [TUR 36] est la théorie sur laquelle sont basés tous nos gadgets électroniques indispensables à la vie des humains du 21ème siècle. Plus abruptement, c'est l'ancêtre de nos ordinateurs et téléphones portables. Bien entendu, ce n'est qu'à la fin de la deuxième guerre mondiale que cette science devient le centre d'intérêt des chercheurs dans le monde entier, et, en moins de 50 ans, l'informatique est devenue l'outil principal poussant les autres disciplines de la science vers l'avant.

Si on parle d'informatique, l'esprit commun pensera directement à l'Intelligence Artificielle (IA). Les fondements de cette discipline remontent aux recherches de Turing [TUR 50] et John McCarthy [MCC 58]. Ce domaine particulier à toujours fait rêver le monde, donnant naissance à un genre littéraire nommé la science-fiction. L'IA prend une place de plus en plus importante dans notre quotidien, comme les moteurs de recherche internet, les applications mobiles, l'industrie automobile et les jeux vidéo, etc.

L'informatique, dans sa globalité s'intéresse au développement d'algorithmes pour automatiser les tâches que les machines doivent effectuer. Un algorithme est un ensemble d'instructions à exécuter successivement pour atteindre un objectif établi [KLE 06]. Cependant, il arrive qu'il soit impossible de trouver un ensemble d'étapes à suivre pour résoudre un problème donné, tel que celui de la prévision Météo. Néanmoins, nous avons accès à une grande quantité d'information sur le sujet étudié. Ainsi, nous avons besoin de méthodes capables d'extraire des connaissances pertinentes ou des patterns répétitifs afin de répondre à la question posée. Ce genre d'algorithmes appartient à la famille des méthodes de fouille de données "Data Mining" (DM) [WAN 03-HAN 12].

La fouille de données est un ensemble de méthodes et outils servant à découvrir des modèles et des connaissances pertinentes à partir de grandes quantités de données. Ces données sont généralement encapsulées en des bases ou des entrepôts de données (Data Warehouse), Web et d'autres référentiels d'informations [HAN 12]. Ces explorations de données sont divisées en deux groupes qui sont les algorithmes descriptifs, et les techniques prédictives. Ces dernières réalisent des inférences sur les données collectées, afin de construire des modèles de prédiction capable de classer si un individu est susceptible d'être intéressé par un produit particulier d'après les informations sur son âge, sexe et ses achats précédents.

La classification est le concept prédictif le plus populaire dans la littérature. Il consiste à établir un modèle ou une fonction permettant de faire la distinction entre des classes se trouvant dans les données considérées ; et ceci à partir d'un ensemble d'apprentissage (le label de la classe est connu) afin de prédire la classe des instances inconnues (ou ensemble de test) [BIS 06-HAN 12-TAN 16]. Ce travail est souvent effectué par un groupe particulier d'algorithmes nommés méthodes d'apprentissage automatique ou artificiel (Machine Learning).

Le croisement entre les principes de l'intelligence artificielle, ceux de la fouille de données, et les méthodes d'apprentissage automatique (Machine learning ML) fournissent aux machines la capacité d'apprendre et d'évoluer au fil du temps, à partir des expériences passées qui se trouvent sous forme de bases de données [COR 11-ALP 14-SHA 14]. Le résultat de l'apprentissage est un modèle de prédiction capable, par exemple, de prédire le temps qu'il fera dans le jour suivant, ou reconnaitre le visage d'une personne filmée, etc.

Le nombre de domaines qui requièrent l'aide des méthodes ML est presque illimité ; par conséquent, chaque champ nécessite une façon différente de penser son algorithme artificiel pour s'adapter au mieux au domaine étudié. Il y a trois types d'algorithmes ML, le premier est l'apprentissage supervisé, où la méthode est complètement contrôlée par un superviseur, en occurrence le programmeur qui établit les directives à suivre à la méthode pour apprendre. Par contre, le deuxième type de systèmes n'a pas besoin d'aide humaine pour fonctionner. Il est appelé méthodes d'apprentissage non-supervisé. Enfin, le dernier groupe de techniques est connu sous le nom d'algorithmes d'apprentissage par renforcement.

Le champ d'applicabilité des systèmes d'apprentissage artificiel est très vaste. Celui qui nous intéresse fortement est le "le génie logiciel". Au premier abord, il très difficile de voir l'utilité des méthodes ML lors de développement d'un programme, où plutôt le rôle que joueraient ces techniques dans le cycle de vie logiciel. Prenons, par exemple, une des phases les plus importantes dans le cycle de développement, et qui est celle des tests logiciels permettant de s'assurer que le système répond aux attentes du client. Pour que le programme développé soit de qualité, le temps et budget que prendrait cette tâche serait très élevé, représentant parfois 50% des ressources allouées [SHE 95-MYE 08-NAI 08-HAS 21]. Donc, une solution s'impose pour essayer de réduire l'effort de test. Une idée est de prédire quelles sont les parties, du système, qui contient des défauts et de ne vérifier que ces parties lors de la phase de test. Ce processus a fait depuis 30 ans l'objet d'études très conséquentes, et a donné naissance au domaine appelé ***Prédiction de défauts logiciels*** (Software Fault Prediction SFP).

Le processus de prédiction de défauts logiciels est chargé par la construction de modèles de prédiction à partir de données sous forme d'historique logiciels, et des informations relatives aux défauts venant d'un ou plusieurs anciens projets, en utilisant le plus souvent des méthodes de classification, plus précisément des algorithmes d'apprentissages artificiels. Afin de déterminer si les entités (classes, méthodes, composants) du nouveau système sont sujettes ou non à contenir des défauts (fault-prone ou not fault-prone) [SHE 95-LES 08-HAL 12-MAL 15-RAT 16-KUM 18-AME 19-ALG 20-PAN 21]. De ce fait, les testeurs ne vont viser que les modules sujets aux fautes, réduisant par la même occasion le temps et le budget alloués à la phase de test, tout en augmentant la qualité du produit [CAT 09-CAT 11].

Il y a trois sous catégories d'études SFP, la plus proéminente d'entre elles est la construction des modèles de prédiction intra-projets, où les données d'apprentissage et de validation sont extraites de la même version du logiciel. Le second scénario de travaux, bien qu'il soit en adéquation avec la définition de la prédiction de défauts logiciels, reste moins exploité dans la communauté SFP. Les modèles inter-projets sont construits à partir de données d'une version antérieure à celle analysée. Durant les dix dernières années, le schéma

Cross-projets, composé de modèles SFP capables de prédire les défauts d'un logiciel qui n'est pas du même projet ou pour un nouveau système, devient de plus en plus populaire où les données d'apprentissages et de tests viennent de deux logiciels complètement différents [ZIM 09-RAT 17-KUM 18-HER 18].

Selon la littérature, les techniques responsables de la construction des modèles de prédiction SFP sont les méthodes d'apprentissage automatiques (ML), et sont plus efficaces que leurs homologues qui sont les techniques de classification statistiques [LES 08-BEE 10-CAT 11-RAD 13-MEL 15]. Un des modèles les plus référencés pour ce genre d'études est le perceptron multicouche (Multilayer Perceptron MLP) qui est un type de réseau neuronal artificiel. Ces réseaux s'inspirent des mécanismes permettant à l'humain d'apprendre et d'évoluer dans son environnement et appartiennent à une classe spéciale de méthodes inspirées des phénomènes biologiques.

La nature a toujours était une source intéressante d'inspiration pour l'homme. L'invention de la roue en est un bon exemple ainsi que les réseaux de neurones artificiels mentionnés ci-dessus. Cette méthode bio-inspirée a depuis longtemps été le centre d'attention des chercheurs en apprentissage automatique, sans oublier ceux de la prédiction de défauts logiciels.

Les systèmes immunitaires artificiels (Artificial Immune Systems ou AIS) correspondent à un nouveau concept qui n'a que trente ans d'existence. Le système immunitaire est responsable de la protection des mammifères de toutes sortes de virus, bactéries et substances chimique dangereuses entravant le bon fonctionnement de l'hôte. Les cellules immunitaires portent des caractéristiques très intéressantes pour développer des techniques d'apprentissage automatique, telles que la mémoire, l'adaptabilité, ainsi qu'un contrôle décentralisé et parallèle. Ce n'est que dans les années 2000 que des tentatives d'implémentation de méthodes d'apprentissage s'inspirant de ce phénomène biologique ont vu le jour [CAR 00-WAT 01-WAT 04].

Parmi les méthodes bio-inspirées, les systèmes AIS sont rarement étudiés dans la littérature SFP, contrairement aux réseaux de neurones, ou aux algorithmes d'intelligence en essaims (Swarm Intelligence). À notre connaissance, seuls cinq travaux ont évalué les systèmes immunitaires artificiels pour la construction des modèles de prédiction de défauts logiciels [CAT 07-CAT 07 A-CAT 09 B-ABE 14-KAU 16]. Néanmoins, quatre de ces travaux ont utilisé le même ensemble de données et sont tous centrés sur la composition des modèles de prédiction intra-projets. Ceci montre à quel point il reste à faire et découvrir à propos de ces systèmes.

Le but de la thèse, comme le titre le suggère, est d'utiliser des méthodes de fouille de données bio-inspirées pour les problèmes liés à l'ingénierie logiciel. Notre choix se porte sur l'étude de la prédiction de défauts logiciels (SFP)en utilisant des méthodes bio-inspirées. Vu que le domaine SFP est assez actif, et que les systèmes AIS sont très peu étudiés dans ce cadre, Il nous a paru intéressant d'étudier l'apport de ce genre de méthodes qui semblent bien adaptées à ce type de problèmes.

Nous allons, dans cette étude, évaluer les performances des systèmes immunitaires pour la construction des modèles SFP sur tous les scénarios possibles. Le premier objectif est de comparer les algorithmes AIS à quelques méthodes d'apprentissage référencées pour la prédiction de défauts logiciels intra-projets (Cross Version CV) en utilisant un ensemble de bases de données publiquement disponibles dans le référentiel de données PROMISE [MEN 16]. Le deuxième objectif est d'évaluer les grandes familles d'algorithme AIS, sur un ensemble de bases afin de confirmer les conclusions auxquelles nous sommes parvenus en atteignant notre premier objectif.

Dans notre démarche, nous tentons aussi d'implémenter un outil d'aide à la prédiction de défauts, vu le manque cruel de travaux s'orientant dans cette direction. Nous souhaitons appliquer notre logiciel à un environnement intra et inter-projets. De ce fait, une étude expérimentale doit être conduite pour vérifier la capacité des AIS à composer des modèles de prédiction SFP Cross-projets afin de mieux ajuster notre système. Ce serait une première étude de ce genre dans le domaine.

Pour atteindre nos objectifs, une large étude empirique doit être menée afin d'évaluer et comparer les systèmes immuno-inspirés pour la construction de modèles SFP inter-projets.

Le reste de cette thèse est organisé comme suit :

Le premier chapitre expose les méthodes de fouille de données bio-inspirées, et plus en détails les systèmes immunologiques automatiques.

Un état de l'art sur le domaine de la prédiction de défauts logiciels est présenté dans le chapitre 2.

Par la suite, le chapitre 3 détaille nos travaux sur les modèles intra-projets, ainsi que la présentation de notre outil d'aide à la prédiction de défaillances logicielles.

Le chapitre 4 représente le fruit de notre travail dans l'étude des modèles inter-projets.

Enfin, une conclusion vient clôturer cette thèse par la présentation des contributions et des éventuelles perspectives de recherches associées.

Chapitre 1 : Méthodes bio-inspirées de Fouille de données

Dans ce chapitre, nous allons discuter des méthodes de Data Mining et nous allons aborder, plus précisément, des techniques auxquelles nous nous intéressons et qui trouvent leur racines dans la biologie, tels que les réseaux de neurones ainsi que les systèmes immunitaires artificiels (Artificial Immune System AIS). Le premier groupe s'inspire des cellules qui forment un réseau à l'intérieur du cerveau humain, et lui donnent la capacité d'analyser et d'évoluer dans son environnement, ainsi que la mémoire pour se rappeler des expériences passées pour apprendre et se développer. Les systèmes immunitaires artificiels encapsulent les agents cellulaires responsables de la protection des humains de leurs environnements remplis de bactéries et de virus, et par conséquent, leur permettre de se prévenir des maladies.

1.1 Introduction

Le monde de l'information, d'internet et des échanges considérables de données électroniques, voilà le résumé du 21ème siècle… une multitude de bases de données prêtes à être utilisées à des fins commerciales ou d'ingénierie, extraites de plusieurs domaines de la vie quotidienne : transactions de vente, Trading, descriptions de produits, expériences scientifiques, observations techniques et surveillance de l'environnement, sans oublier les réseaux sociaux [HAN 12]. Dans le dessein d'exploiter ces données et trouver une information utile (knowledge) ou un pattern sous-jacent, nous avons besoin d'outils d'analyse puissants afin de minimiser les erreurs, ainsi que la perte considérable de temps et de ressources. Ces outils appartiennent à la famille de méthodes de fouille de données (Data Mining) [WAN 03-HAN 12-HAN 15- TAN 16] qui pourrait être définie comme suit :

« **Data Mining (DM):** Un ensemble de méthodes et outils servant à découvrir des modèles et des connaissances pertinentes à partir de grandes quantités de données. Ces données sont généralement encapsulées en bases de données, entrepôts de données (data Warehouse), Web et d'autres référentiels d'informations… [HAN12].»

Les méthodes d'exploration de données peuvent être divisées en deux groupes distincts (*Figure 1*.1) : Les méthodes **descriptives**, et les techniques **prédictives**. Ces dernières réalisent une induction sur les données étudiées afin de faire des prédictions, comme classer les personnes susceptibles d'avoir la grippe selon leur lieu de résidence. Alors que les méthodes descriptives se centrent sur la découverte des relations et les patterns qui peuvent relier les instances d'une base de données étudiée, exposant ainsi ses propriétés [WAN 03].

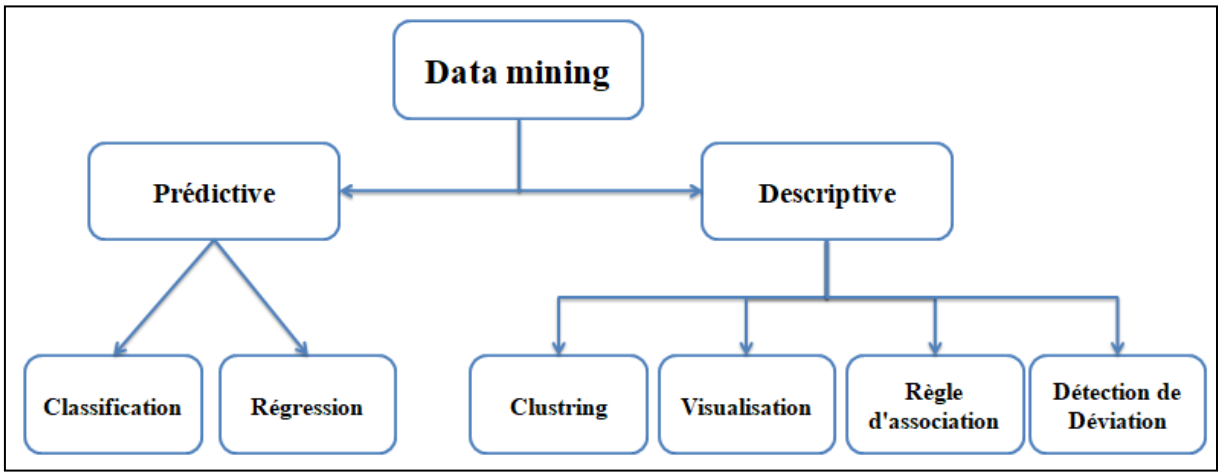


*Figure 1.*1-Taxonomie des méthodes de fouille des donnés

La méthode la plus populaire dans l'exploration de données est **la classification**, où le but est de trouver un modèle ou une fonction permettant la distinction (ou discrimination) d'un ou plusieurs concepts appelés **classes,** se trouvant dans les données, cela à partir d'un ensemble d'apprentissage (dans lequel l'étiquette de la classe est connue) afin de prédire la classe des instances inconnues (ou ensemble de test) [BIS 06-HAN 12-TAN 16]. Le modèle construit peut se présenter sous plusieurs formes, telles qu'un ensemble de règle If-Then, une formule mathématique, un arbre de décision ou un réseau de neurones [HAN 12]. Ces représentations sont les résultats d'un groupe particulier d'algorithmes, nommées techniques **d'apprentissage automatique** (Machine Learning).

1.2 Apprentissage automatique (Machine Learning)

Un algorithme est une suite finie d'opérations ordonnées devant être suivies pour résoudre un problème donné [KLE 06]. Le rôle d'un ingénieur en informatique est de trouver l'algorithme adéquat pour accomplir une certaine tâche telle que le tri, la recherche ou la cryptographie…

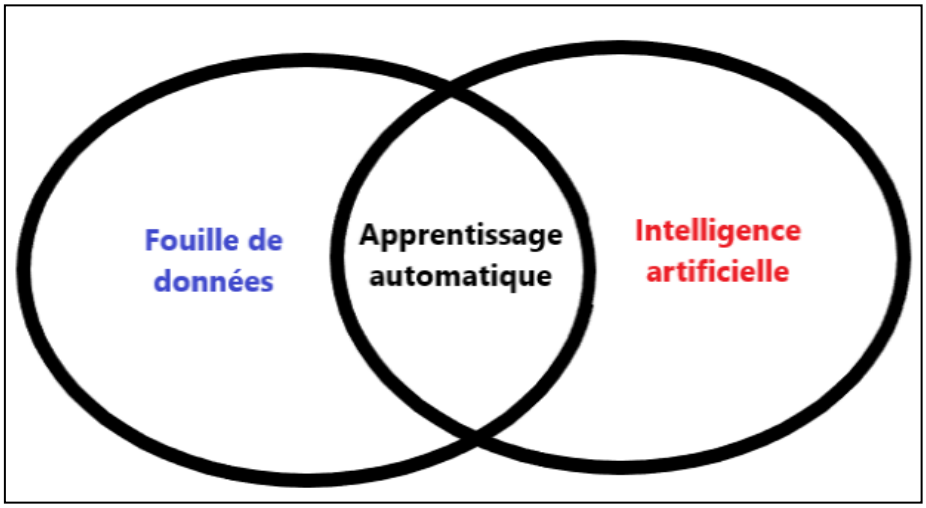


*Figure 1.*2-Lien entre le Data mining et Intelligence artificielle

Il s'avère parfois difficile, ou même impossible, de trouver une solution algorithmique à notre problème. C'est le cas, par exemple pour la reconnaissance des visages, la détection de cellules cancéreuses ou la reconnaissance de l'écriture manuscrite, par contre, nous avons assez de données qui résultent d'enregistrements passés, tels que des images médicales ou des photos familiales, qui peuvent nous donner une issue approximative à notre problème. De ce fait, nous aurons besoin d'algorithmes capable d'extraire des connaissances pertinentes à partir de ces données brutes, ces algorithmes sont le résultat du mariage entre **l'intelligence artificielle (IA)** et le **Data mining (DM)** qui sont les méthodes **d'apprentissage automatique (Machine learning)** [ALP 14] (*Figure 1.*2).

Apprendre à partir de l'expérience passée, représentée sous forme de base de données est le rôle des méthodes d'apprentissage automatique, qui donnent ainsi aux machines la capacité d'apprendre et d'évoluer, au fil du temps, dans leur environnement [COR 11-ALP 14-SHA 14].

Concrètement, un ensemble *S* de données, appelé **ensemble d'entrainement** (ou d'apprentissage) est présenté à l'algorithme d'apprentissage, ce qui va permettre à la méthode d'extraire, par inférence, une connaissance sous forme de modèle. Cce modèle est une fonction, qui va prédire la classe inconnue des instances non étiquetées constituant **ensemble de test** ; c'est ce que nous appelons une **généralisation** [BIS 06-COR 11]. En outre, si le résultat de la généralisation est un ensemble discret de cas, comme la reconnaissance des lettres de l'alphabet, il s'agit de **classification.** D'un autre côté, si les sorties sont une ou plusieurs variables continues, comme la prédiction du nombre de jours pour la construction d'un immeuble, là nous nous intéressons à la **régression** [BIS 06-SHA 14].

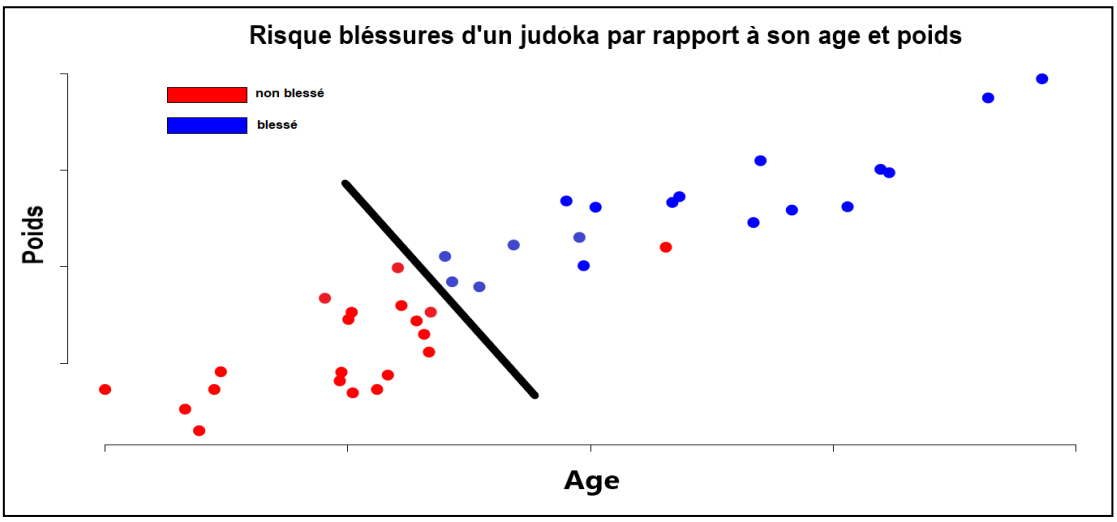


*Figure 1.*3-Exemple d'un résultat d'une méthode d'apprentissage

Exemple : « Dans le but d'évaluer le risque qu'un judoka[1] subisse une blessure lors d'une compétition, une méthode d'apprentissage artificiel a été appliquée sur une base

[1] Le judo est un art martial japonais crée par Jigorō Kanō en 1882, appelé aussi la voie de la souplesse, c'est le sport favori de l'auteur de la thèse

d’apprentissage, où ses instances sont représenté par leur âge ainsi que leurs poids, séparé en deux classe, la première étiquette (non blessé) informe que le judoka n'a jamais été blessé, alors que la deuxième, indique que le sportif a été déjà blessé dans sa carrière. Le modèle appris par l’algorithme est illustré par la *Figure 1*.3, où la connaissance extraite à partir des donnés, nous renseigner que : « plus le judoka est âgé, plus il a de risque de se blesser ».

Le nombre de domaines qui requièrent l'aide des méthodes d’apprentissage est presque illimité, par conséquent, il existe différentes approches d’apprentissage artificiel pour s'adapter au mieux au problème étudié. En effet, l'implémentation d'une technique d'apprentissage automatique peut être effectué selon l’un des trois paradigmes suivants (*Figure 1*.4) [BIS 06 –COR 11-ALP 14]:

L'apprentissage supervisé (Supervised Learning) : Dans ce paradigme, les exemples d'apprentissage sont étiquetés afin d'identifier la classe à laquelle ils appartiennent. Le but de l'algorithme est de classifier correctement les nouveaux exemples dans les classes définies dans la phase d'apprentissage. Comme l’illustre notre exemple précédent (*Figure 1*.3).

L'apprentissage non supervisé (Unsupervised Learning ou clustering) : le principe est de diviser un groupe hétérogène de données, en sous-groupes de manière que les données considérées comme les plus similaires soient associées au sein d'un groupe homogène appelé cluster, par contre les données considérées comme différentes se retrouvent dans d'autres clusters distincts, sans intervention d’un superviseur. Ce paradigme est généralement utilisé pour la détection des anomalies (une instance qui n’appartient à aucun des clusters construit par l’algorithme).

L’apprentissage par renforcement (**Reinforcement Learning**) : ce mode d’apprentissage suit un processus stochastique pour trouver la séquence d’actions appropriées à effectuer pour que l'agent apprenant change d'état. Contrairement à l'apprentissage supervisé, l'apprenant ne reçoit pas d'exemples de sorties optimales, mais doit plutôt découvrir, par un processus d'essais/erreurs, le meilleur schéma à suivre pour accomplir une tache donné, tout en maximisant les récompenses fournies par le superviseur. Un robot qui cherche à trouver sa sortie d'une chambre, à chaque pas au hasard (une action) le robot change d'état, s'il est proche de la sortie par rapport à l'état précédent, il reçoit une récompense jusqu'à trouver le bon chemin à suivre pour quitter la pièce (maximiser le nombre de récompenses).

L'inspiration humaine trouve souvent son origine dans la nature et les interactions biologiques, c’est notamment le cas pour l'informatique et, plus précisément, l’apprentissage automatique. L’outil principale qui permet à l'humain d’apprendre et mémoriser n'étant autre que le cerveau, l'humain va chercher à faire reproduire les mécanismes d’apprentissage et de mémorisation par les ordinateurs, d'où la naissance des réseaux de neurones artificiels que nous allons aborder dans la section 1.3. Un autre phénomène intéressant dans le corps humain est la capacité de se protéger des maladies et de prospérer dans son environnement, parfois dangereux. Le système immunitaire est responsable de cette protection et constitue, depuis plusieurs années, une nouvelle source d’inspiration passionnante pour les chercheurs en apprentissage automatique. Ceci a donné naissance aux Systèmes Immunitaires Artificiel

(AIS) qui représentent le centre d'intérêt de notre thèse, et que nous allons essayer de présenter dans la section 1.4.

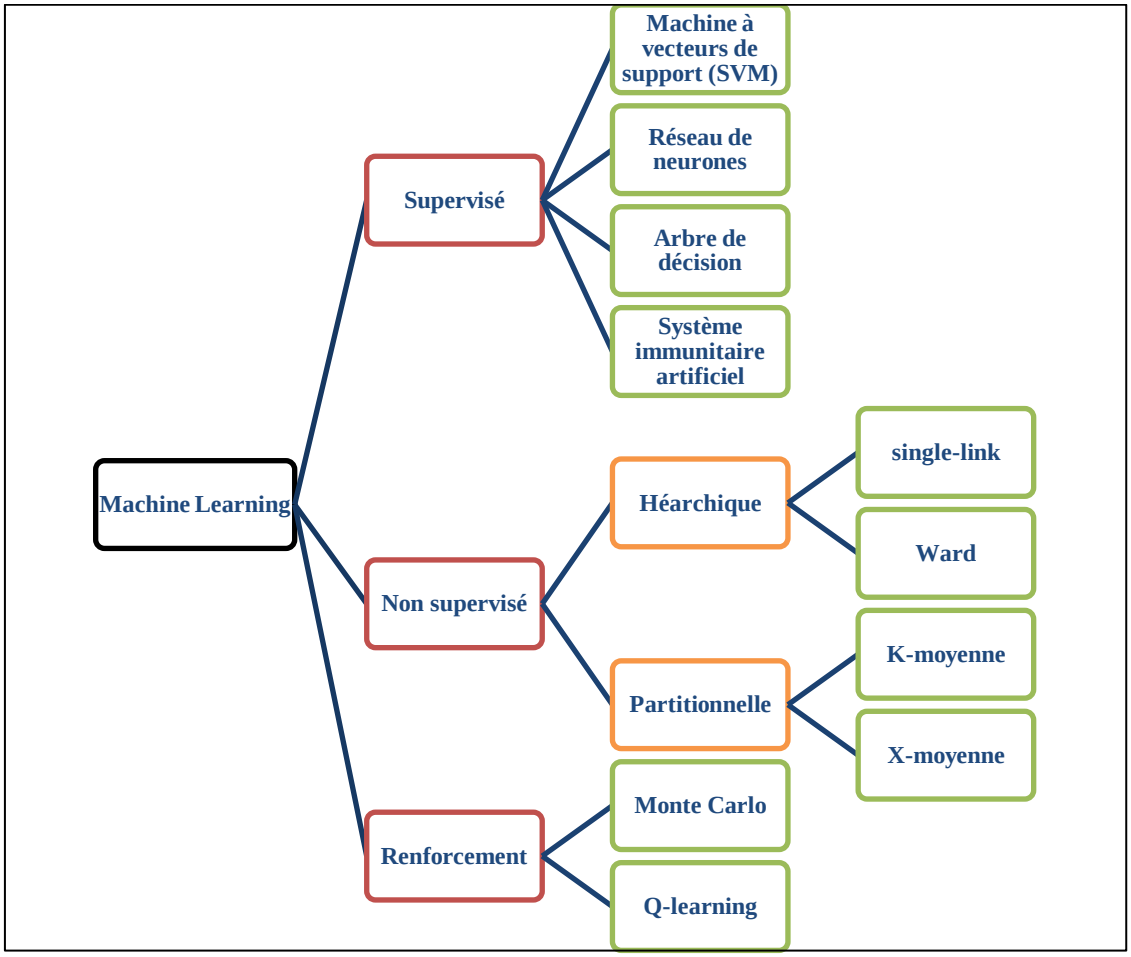


Figure 1.4-Types d'algorithmes d'apprentissage automatique

1.3 Les réseaux de neurones artificiels

Le cerveau humain est une incroyable machine constituée de plusieurs milliards de **neurones** interconnectés (*Figure 1.*5), lui donnant la capacité d'apprendre, reconnaitre et s'adapter à son environnement. Ainsi, c'est devenu une grande source d'inspiration pour les chercheurs en IA, qui veulent implémenter et automatiser ces tâches quotidiennes. Une tâche telle que la reconnaissance d'un nombre écrit à la main est presque intuitive pour l'homme. Par contre, écrire un algorithme spécifique pour doter l'ordinateur d'une telle fonctionnalité est presque impossible, car l'humain lui-même est incapable de décrire comment il arrive à traiter et mémoriser les informations qui lui parviennent, même s'il connait l'entité responsable de ce processus [NIE 15].

Même s'il est admis qu'un neurone est moins rapide et plus simple qu'un processeur, la puissance du cerveau lui vient de son inter-connectivité qui lie un très large ensemble de neurones au niveau des **synapses** [ALP 14]. Ainsi, le cerveau est capable de traiter l'information d'une façon parallèle et distribuer la mémoire d'une manière passive dans tout le

réseau neuronal. D'une manière plus simple, le **noyau** est considéré comme responsable des tâches dites intelligentes, alors que la mémoire se trouve dans les synapses qui lient les neurones au niveau des **dendrites** et des **axones [**COR 11-NIE 15].

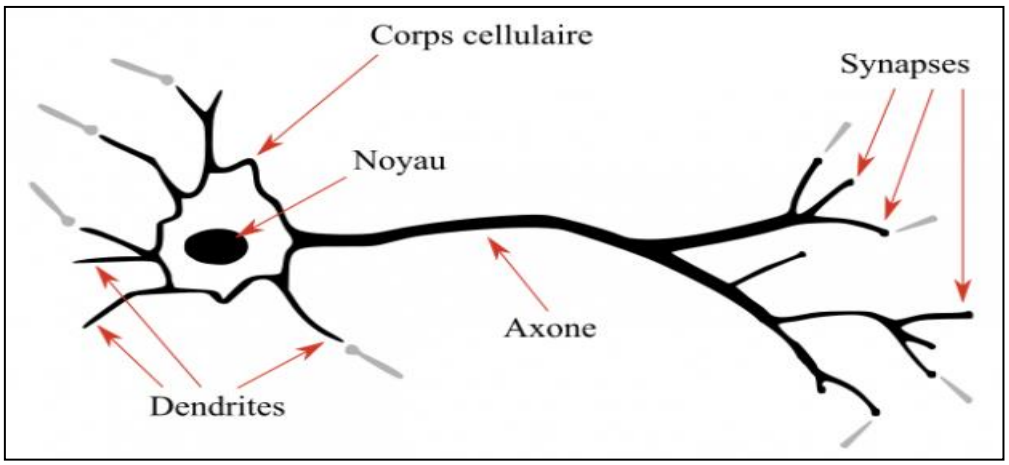


Figure 1.5 **Les composants d'un neurone biologique [1]**

Au début des années 60, ces éléments ont donné naissance aux méthodes les plus populaires da l'apprentissage automatique qui est **les réseaux de neurones atrificiels (Artificial neural networks),** composé de plusieurs neurones reliés entre eux.

1.3.1 L'algorithme du Perceptron :

Le Perceptron (*Figure 1*.6) est considéré comme le premier modèle d'apprentissage automatique introduit Frank Rosenblatt [ROS 61], c'est la représentation la plus simple d'un réseau de neuronal (ANN) [KLE 06-NIE 15]. L'algorithme du Perceptron simule le travail d'un seul neurone, il a en amont un ou plusieurs entrées $X_i = \{x_1, x_2, x_3, \ldots x_n\}$ et chacune d'elles est associée à un poids $W_i = \{w_1, w_2, w_3, \ldots w_n\}$. Dans le cas de notre exemple vu dans la section 1.2, x_1 représente l'âge et x_2 le poids d'un judoka, ajouter à cela la connexion biaisé x_0 où l'entrée est toujours initialisée à 0 et son poids $w_0=1$ afin de généraliser l'algorithme et s'éloigner du point d'origine [ALP 14-NIE 15].

Dans le cas le plus simple **la fonction d'activation** du neurone est exprimée dans l'équation 1.1 :

$$Y=\sum_{i=0}^{n} x_i w_i + w_0 \quad (1.1)$$

Lors de la phase d'apprentissage, l'algorithme doit trouver l'ensemble idéal des poids *Wi* afin que le résultat prédit corresponde avec la classe fournie par le superviseur à l'entrée, à la fin de la période d'entraînement. Le Perceptron va dessiner un hyperplan (*Figure 1*.3) séparant les judokas en 2 classes distinctes (blessé /non blessé). Le résultat de la prédiction pour chaque entrée est pourvu par la fonction seuil ci-dessous :

$$Classe = \begin{cases} 1, & y > seuil \\ 0 & \end{cases} \quad (1.2)$$

Où 1 représente la classe blessé, et 0 les non blessés pour le cas de l'exemple 1. Les poids *Wi* sont mis à jour afin de réduire l'erreur de classification grâce à l'équation suivante :

$$Wi + 1 = Wi + \alpha(C - P)Xi \quad (1.3)$$

Où *Wi+1* est le nouveaux poids, α est le taux d'apprentissage, tandis que c'est la classe attendue alors que P est la classe prédite, est *Xi* est l'entrée liée avec le poids *Wi*. L'erreur de classification est donnée par :

$$E = \sum_{i=1}^{k}(C - P_k)(1.4)$$

Où K représente une instance de la base de données (Poids/ Age d'un judoka).

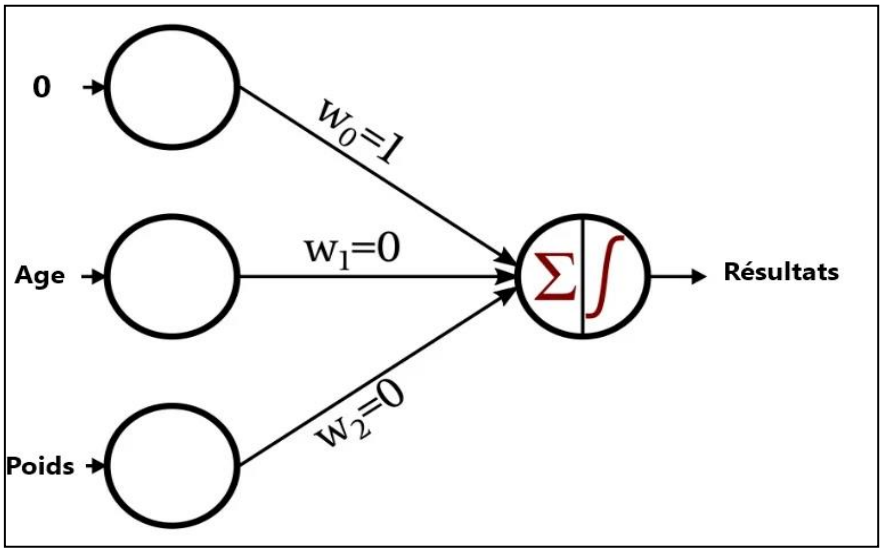


Figure 1.6- Le schéma d'un Perceptron

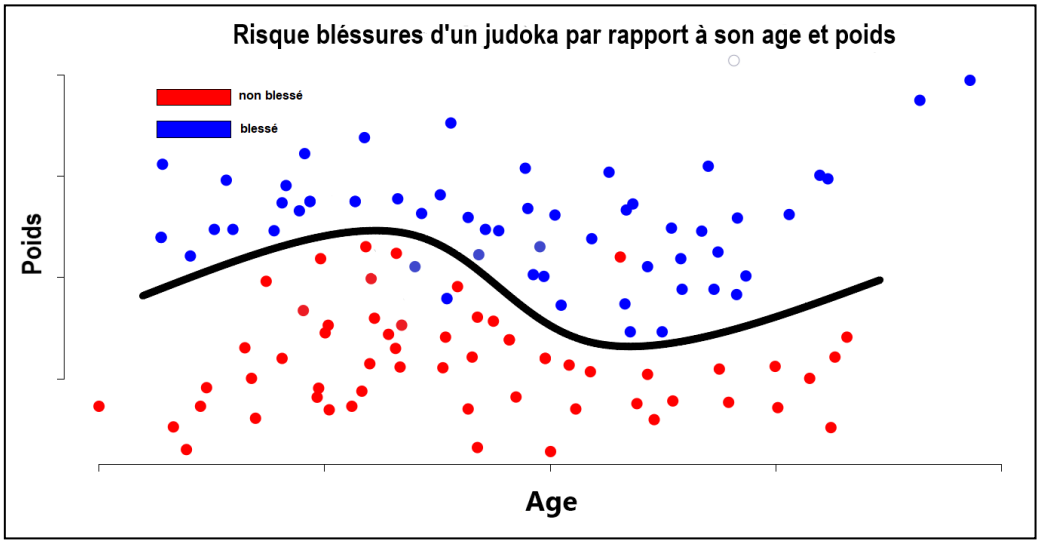


Figure 1.7- Exemple d'un résultat non linéairement séparable

L'algorithme du Perceptron est une méthode d'apprentissage linéaire, ceci veut dire qu'il ne s'intéresse qu'aux problèmes linéairement séparables [KLE 06-COR11-ALP14-NIE 15], où nous pouvons dessiner une simple droite capable de séparer les instances d'une base de données étudiée comme dans la *Figure 1*.3.

Si nous prenons notre exemple présenté dans la *Figure 1*.3 et ajoutons plus d'instances à notre base pour mieux généraliser notre méthode d'apprentissage, il devient impossible à l'algorithme du perceptron de résoudre le problème car nous ne pourrons plus dessiner une simple droite pour séparer les judokas blessés des autres comme le montre la *Figure 1*.7. Une nouvelle méthode plus puissante s'impose qui n'est autre que celle du **Perceptron multicouches (Multilayer Perceptron ou Back Propagation Neural Network).**

1.3.2 Perceptron Multicouches/Multilayer Perceptron (MLP):

Dans la version la plus simple du Perceptron Multicouches, les réseaux de neurones contiennent au moins trois couches de nœuds (*Figure 1*.8), chaque neurone d'une couche adjacente est lié de manière pondérée à tous les nœuds du niveau précédent (*Wi*), ainsi qu'un biais b0 représentant la difficulté de l'activation des neurones (firing of the neurone) [ALP 14-NIE 15]. Un tel réseau de neurones est théoriquement capable de simuler n'importe quelle fonction non-linéaire [KLE 06].

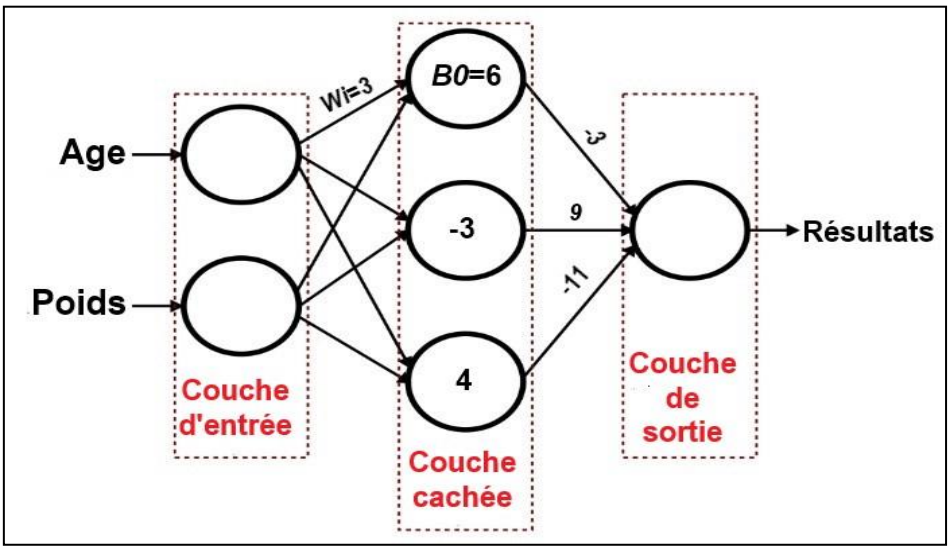


Figure 1.8- Le schéma d'un Perceptron Multicouche

Lors de la période d'apprentissage, le but de l'algorithme est de trouver les poids *Wi* ainsi que les biais *b0* idéaux afin de classifier les instances d'une base de données correctement. Cela se fait grâce la minimisation de la fonction d'erreur quadratique suivante :

$$E = \frac{1}{2}\sum_{i=1}^{n}(C - P)^2 \quad (1.5)$$

Où, C'est la classe attendue alors que P est la classe prédite par le réseau, la recherche des paramètres (poids/biais) est induite par la méthode de la **descente du gradient.**

Pour chaque instance de la base de données (dans notre exemple, le couple poids et âge munis de sa classe C), nous la faisons passer par le réseau d'une façon directe (de l'entrée vers la sortie) et la valeur d'activation d'un neurone est calculée comme suit :

$$Y = f\left(\sum_{i=1}^{n} x_i w_i + b_0\right) \quad (1.6)$$

Où, *xi* est l'entrée liée au poids *wi* pour un nœud dans la couche cachée, et *f* est la fonction d'activation non linéaire, généralement la fonction *Sigmoïde :*

$$\boldsymbol{f(z) = \frac{1}{1+e^{-z}}} \quad \textbf{(1.7)}$$

Il y a d'autres fonctions d'activation que celle présenté dans 1.7, mais dans notre thèse nous avons utilisé qu'elle, de plus c'est la procédure la plus populaire dans le développement d'un réseau de neurone classique de type Perceptron Multicouches.

Après la phase d'activation des nœuds de la couche cachée, c'est le tour des neurones de la couche de sortie. Dans le cas où la sortie correspond à la classe étiquetée par le superviseur nous passons l'entrée suivante, sinon nous minimisons la fonction d'erreur *E (1.5)*. Pour cela, nous faisons un retour arrière (de la sortie vers l'entrée) en modifiant les poids et biais par la méthode de la descente du gradient (d'où l'appellation Rétropropagation du gradient ou **Backpropagation neural network**), les nouveaux poids et biais sont calculés comme suit :

$$\boldsymbol{W(n+1) = W(n) - \alpha(\frac{\partial E}{\partial w(n)})} \quad \textbf{(1.8)}$$

$$\boldsymbol{b(n+1) = b(n) - \alpha(\frac{\partial E}{\partial b(n)})} \quad \textbf{(1.9)}$$

Où *W (n+1) et B(n+1)* sont les nouveau poids et biais du nœud *n, α* est le taux d'apprentissage et $\frac{\partial E}{\partial v(n)}$ est la dérivée partielle de la fonction *1.5* par rapport à un des paramètres que nous voulons minimiser. La solution de la dérivée est hors de portée du sujet de la thèse. Nous vous dirigeons à la vidéo explicative de **Josh Starmer**[2] dans la plateforme **Youtube** pour avoir une vision simplifiée de comment est faite la recherche des nouveaux poids et biais dans le réseau de neurone en utilisant la méthode de la rétropropagation du gradient.

Avant de terminer cette section une petite parenthèse s'impose, à cette époque tout le monde ou presque à entendu parler du **Deep learning**. De manière simplifier le Deep learning n'est qu'une généralisation du MLP, où le but est d'avoir un grand nombre de couches cachée d'où la formulation Deep, un autre changement, c'est la fonction d'activation, cette fois-ci, c'est la méthode **ReLU (rectified linear unit)** " ***f(z)=max(0,z)***"qui est utilisé, car elle est moins lourde et gourmande que l'équation 1.7 et a fait ses preuves surtout dans la reconnaissance et le traitement d'images [ALP 14-NIE 15-BUR 20].

Dans la suite de ce chapitre, nous allons discuter d'autres méthodes bio-inspirées moins connues et relativement nouvelles dans la scène de l'apprentissage artificiel, ce sont les **Systèmes Immunitaires Artificiels (AIS)** qui s'inspirent du système responsable de la protection des mammifères contre les maladies**.**

[2] https://www.youtube.com/c/joshstarmer/featured

1.4 Système Immunitaire Artificiel « Artificial Immune System (AIS). »

Le système immunitaire est doté de plusieurs caractéristiques recherchées dans le domaine du ML, le rendant une excellente source d'inspiration pour les chercheurs, nous pouvons résumer ses caractéristiques comme suit [HOF 99]:

- **Variabilité (Diversité) :** Différentes personnes sont vulnérables à différents microbes améliorant ainsi sa robustesse.
- **Distribution (parallélisme)** : il n'y a pas de contrôle central et donc pas de point de défaillance unique.
- **Tolérance aux erreurs** : les erreurs n'ont pas d'incidences catastrophiques pour l'hôte.
- **Dynamicité :** Ses composants sont continuellement créés, détruits et circulent dans tout le corps, lui permettant de se débarrasser des éléments inutiles ou dangereux pour l'hôte, tout en améliorant les composants existants.
- **Auto-protection** : les mêmes mécanismes qui protègent le corps protègent également le système immunitaire lui-même.
- **Adaptabilité** : Ses mécanismes lui permette d'apprendre à reconnaître et à répondre facilement à de nouveaux microbes.
- **Mémoire :** les systèmes conserve une mémoire des antigènes infectant le corps, pour faciliter l'élimination lors d'une réinfection par les mêmes antigènes responsable du premier mal.

Avant de d'aborder la description des méthodes immunitaires artificielles, la suite de cette section va présenter les éléments et les mécanismes du système naturel, qui ont influencés la création de ces techniques.

1.4.1 Les systèmes immunitaires naturels

Le système immunitaire représente un mécanisme d'identification capable de combattre le dysfonctionnement de ses propres cellules et les micro-organismes étrangers tels que les bactéries, virus ou tout autre agent infectieux qui envahissent le corps à longueur de journée. C'est une collection d'organes, cellules et molécules travaillant ensemble pour garantir le développement et l'adaptation de l'hôte dans son environnement [DAS 09, DEC 99].

La fonction primaire de ce système est la distinction entre le soi (toute molécule appartenant au corps) et le non-soi (les antigènes), donc il doit répondre uniquement aux entités étrangères qui causent son activation [GHA 06, SAI 11]. Si le système immunitaire n'est pas capable d'effectuer cette distinction, alors une réponse immunitaire sera déclenchée contre le soi, provoquant des maladies auto-immunes [DAS 09]. Nous pouvons résumer le soi et le non-soi comme suit [2] :

- **Le soi** : un ensemble de protéines résultant de l'expression du génome d'un individu (appelé CMH) qui va représenter son identité pour le système, par conséquent uniques

à chaque entité, ces marqueurs cellulaires chez les humains sont appelés système HLA (Human Leucocyte Antigen).

- **Le non soi :** c'est un ensemble de molécule différentes du soi, si elles sont présentes dans l'organisme, elles déclencheraient des réactions immunitaires. Elles peuvent être issues du milieu extérieur (vers, virus, bactéries, toxines…) ou être simplement des molécules du soi modifiés comme dans le cas des cancers.

Nous pouvons voir le système immunitaire comme étant un réseau multicouche [DAS 09], la première couche représente l'immunité innée et le deuxième appelé immunité acquise spécifique ou adaptative. **L'immunité innée est** présente dès la naissance, elle agit sans tenir compte du type de l'entité qu'elle combat, de plus elle effectue une distinction globale du soi et du non-soi toute en agissent d'une manière immédiate est non adaptative. Enfin, elle initialise et régularise la réponse immunitaire adaptative. Les éléments responsables de cette réponse sont [DAS 09]:

- Les cellules NK, les phagocytes et les macrophages...
- Les conditions physiologiques telles que le pH et la température du corps.
- La peau.
- Le système respiratoire.
- La salive, et toute sorte de mucus.

L'immunité acquise (adaptative) ou spécifique [DAS 09] est une réponse où la mémoire joue un rôle prépondérant pour l'élimination des pathogènes, elle se base sur la reconnaissance grâce au Paratope des antigènes à combattre et ensuite mémoriser cet événement. Nous pouvons distinguer deux types de réponses adaptatives [BRO 11] :

- Réponse primaire qui se produit lorsque le système rencontre un nouveau pathogène. Le système serait lent à réagir, prenant ainsi plusieurs semaines pour éliminer l'infection. À la fin de cette étape, le système va mémoriser les caractéristiques de l'antigène (les molécules du non soi).
- Réponse secondaire qui s'applique lorsque le même agent pathogène réinfecte le corps, le système élimine rapidement l'infection en utilisant ce qui a été appris dans la réponse primaire. La mémoire que le système acquiert dans la réponse primaire est généralement de longue durée, fournissant une immunité contre le pathogène pour la durée de vie de l'hôte.

Les systèmes immunitaires naturels sont composés de plusieurs éléments travaillant en symbiose afin d'éliminer tout risque qui peut atteindre l'entité où ils se trouvent, ces éléments sont comme suit [GHA 06, DAS 09, DEC 99] :

Les organes : Nous pouvons les diviser en deux classes (*Figure 1.*9) :

1) *Les organes lymphoïdes primaires :* ils constituent le site de développement et de maturation des cellules :
   a) Moelle osseuse : Lieu de développement des cellules progénitrices lymphoïdes qui après leur croissance se diviseront pour former les cellules précurseurs des lymphocytes B et T.
   b) *Thymus* : Dans le bas du cou, constitue le site de maturation des lymphocytes T, mais la plupart d'entre eux meurent sur place, seul 5 % quittent le thymus.
   c) *Vaisseaux lymphatiques* : Transportent la lymphe, les vaisseaux lymphatiques sont situés dans tout le corps.
2) *Les organes lymphoïdes secondaires :* ils constituent le lieu d'interaction entre l'antigène et le lymphocyte.
   a) Les *amygdales.*
   b) Les *ganglions lymphatiques.*
   c) La rate.

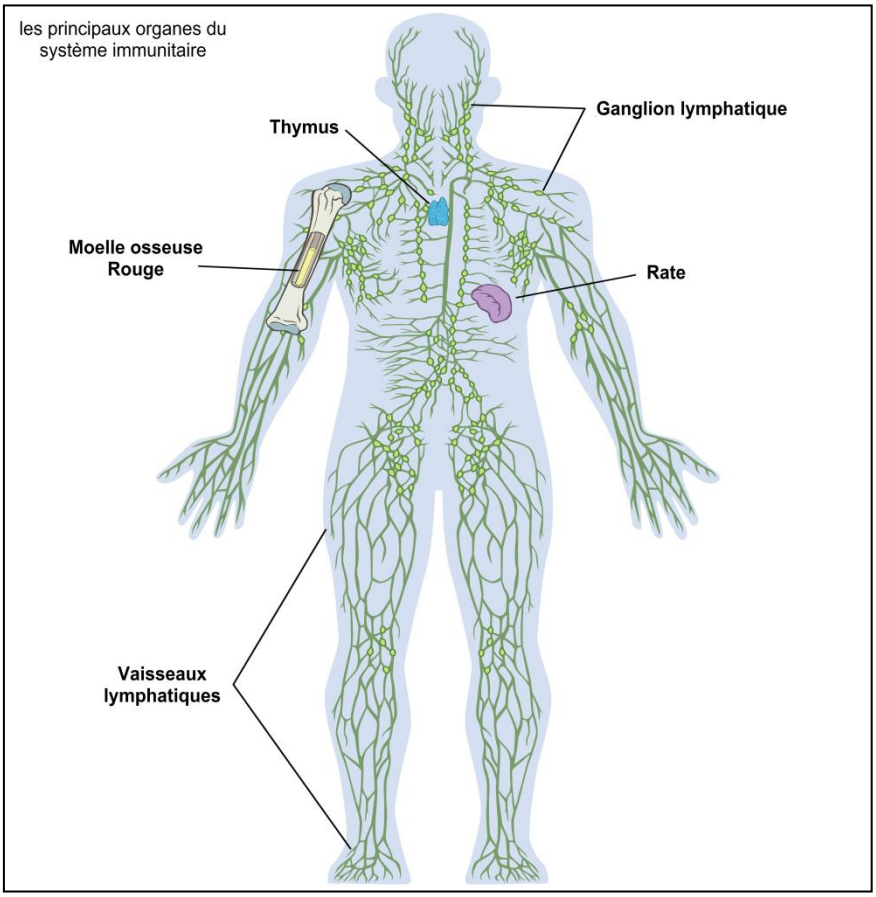


***Figure 1.9*- Les principaux organes de la réponse immunitaire [3]**

Les cellules : Les globules blancs aussi appelés leucocytes qui se trouvent dans le sang, la lymphe et les organes lymphoïdes sont les cellules responsables de protéger le corps des organismes étrangers. Il existe différents types de leucocytes (*Figure 1*.10), dont les plus importants sont lymphocytes B et T.

Lymphocytes T : Il existe plusieurs types de lymphocytes T :

- Les lymphocytes *T cytotoxiques (Tc)* qui, lorsqu'elles sont activées, détruisent directement les cellules infectées par des virus et les cellules tumorales. Alors que les *T auxiliaires* se chargent de la coordination des autres aspects de la réponse immunitaire en déclenchant déclenchent à la fois le clonage et par la même occasion la stimulation ou suppression des anticorps secrétés par les cellules B.
- Les lymphocytes T *suppresseurs* sont des régulateurs de l'immunité et luttent contre les réactions auto-immunes.

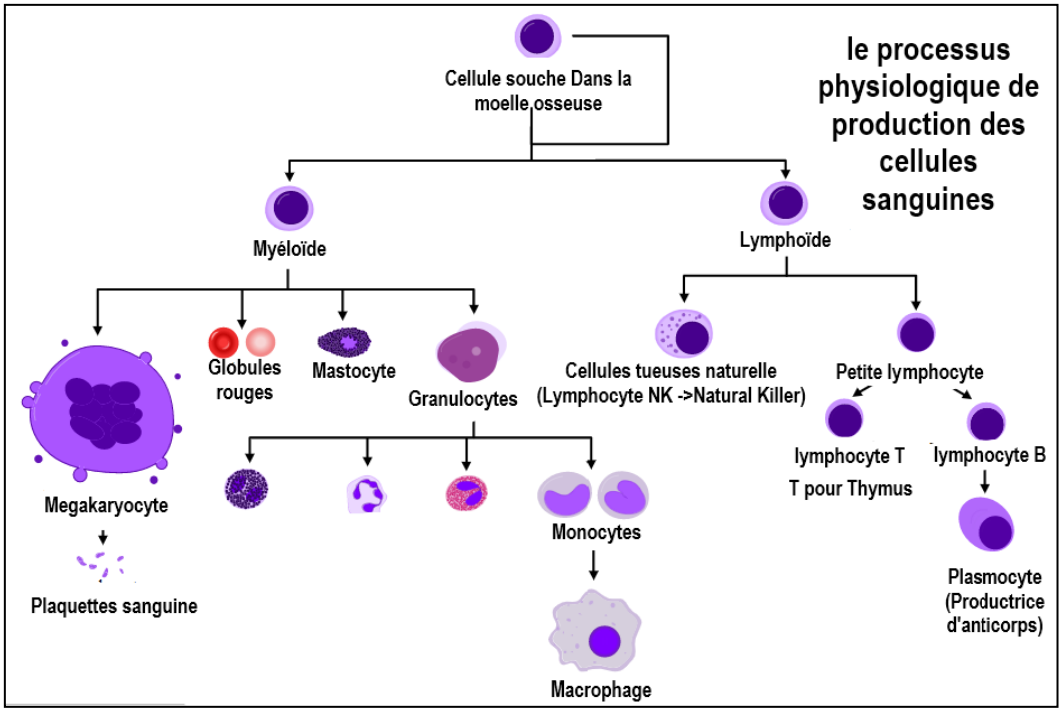


Figure 1.10- Les cellules responsables de la réponse immunitaire[3]

Lymphocytes B : la fonction principale des lymphocytes B est de produire des protéines appelé anticorps qui se fixent sur les protéines étrangères des antigènes (le non-soi). La partie de l'anticorps responsable de la reconnaissance de l'antigène est appelée *paratope*. Le

[3] https://en.wikipedia.org/wiki/Haematopoiesis

paratope se lie à une partie spécifique de l'antigène appelée *épitope* (*Figure 1.*11). La liaison entre un paratope et un épitope est d'autant plus forte que leurs formes sont complémentaires. La force de cette liaison est appelée *affinité* [DAS 09-BRO 11]. Les cellules B qui vont mieux reconnaître l'antigène vont proliférer en se clonant. De plus, les cellules B ont également la capacité de se comporter en cellule mémoire d'un antigène déjà combattu par le système afin de l'éliminer efficacement la prochaine fois qu'il entre dans le corps (adaptation).

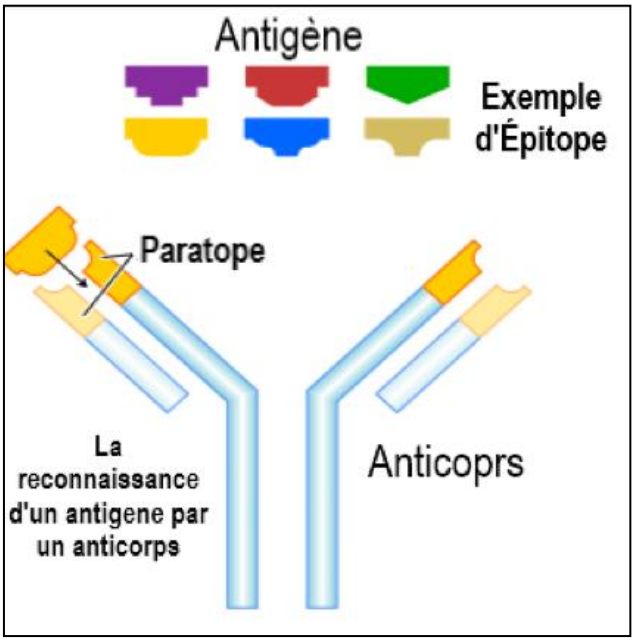


*Figure 1.*11- **Exemple schématique de la reconnaissance entre anticorps et antigène**

Les cellules NK (tueuse) : qui n'expriment ni les récepteurs des B ni ceux des T. Elles ne sont pas spécifiques et n'ont pas de mémoire.

1.4.2 Les systèmes immunitaires artificiels

La naissance des méthodes AIS a eu lieu grâce aux travaux de Farmer et al dans leur article "The immune system, adaptation and machine learning" [FAR 86] et celui de Bersini et de Varela sur les réseaux immunitaires en 1991 [BER 91]. Cependant, ce n'est qu'au milieu des années 90 que les AIS sont devenus une discipline de recherche à part entière. Dans l'année 1994, Dasgupta a entrepris des études étendues sur des algorithmes de sélection négative. Hunt et Cooke ont commencé des travaux sur les modèles de réseaux immunitaires en 1995, Timmis et Neal ont continué ce travail et ont apporté quelques améliorations [TIM 02]. Le travail de De Castro, Von Zuben, Nicosia et de Cutello sur la sélection clonale est devenu une référence en 2002 [CUT 02]. Le premier livre sur les systèmes immunitaires artificiels a été publié par Dasgupta en 1999 [DES 99]. Notons aussi les travaux de Watkins sur un algorithme d'apprentissage automatique inspiré des phénomènes immunologiques dans sa

thèse de master en 2001 [WAT 01] et ceux de Carter [CAR 00] en 2000, qui a introduit le premier algorithme d'apprentissage automatique supervisé.

Le rôle des méthodes AIS n'est pas de reproduire les phénomènes qui entourent leur inspiration car cela serait trop complexe à implémenter [HOF 99], mais plutôt tirer parties des avantages et fonctionnalités qu'incarne le système naturel, comme la mémoire, l'adaptation, le parallélisme [BER 91-TIM 02] ...

Pour faire simple, le but des méthodes immunitaires est de créer une cellule mémoire qui va représenter la solution à un problème donné [DAS 99], qu'il soit question d'apprentissage automatique, clustering, optimisation ou détection d'intrusion, les principaux acteurs des méthodes immunologique et leurs rôles dans l'apprentissage sont[4] :

- **Antigène (Ag) :** il représente l'entité à apprendre par le système, ou simplement les instances d'une base de données. *Ag* est un vecteur de caractéristiques *Ag ={a ,b ... ,n}*. où a, b et n sont des variables indépendante représentant l'épitope de l'antigène. Si nous reprenons l'exemple qui se trouve dans la section 1.2 *Ag={age, poids}*. Dans le cas de l'apprentissage supervisé, l'antigène sera muni d'une classe *C* représentant la variable dépendante par exemple *Ag.c {blessé}*.
- **Anticorps/Antibody (Ab) :** ils servent à reconnaître et éliminer les antigènes, simplement les entités apprises par le système. *Ab* est un vecteur de la même dimension que *Ag*, ils peuvent être clonée et muté pour mieux correspondre aux instances de la base de données est avoir un modèle d'apprentissage plus robuste.
- **Cellule T/T-cell** : les cellules responsables de l'identification primaire de l'antigène (épitope) dès qu'il entre dans le corps, ainsi qu'à la stimulation des cellules B pour la production spécifique des anticorps. L'indentification de l'Ag par la T-cell se fait grâce une fonction de distance *(d)* entre leurs vecteurs, plus d est petite plus l'algorithme arrive à reconnaitre l'antigène. Généralement la distance euclidienne est utilisée.
- **Cellule B/B-cell** : Après avoir été stimulée par la T-cell, elle produit les anticorps nécessaires à l'élimination de l'object étranger. Les B-cell sont aussi responsable du clonage et mutation des *Ab* afin de mieux assimiler le problème. A la fin les

[4] Les rôles ici sont liés à l'apprentissage automatique, donc les termes cités sont propres au Machine Learning

cellules B sont sauvegardées en tant que cellule mémoire et une solution du problème d'apprentissage.

- **Cellule mémoire/Memory cell (Mc) :** généralement des B-cell après avoir éliminé l'antigène, gardent les informations apprises durant la réponse immunologique, dans le cas d'une nouvelle infection par le même *Ag* le système le rejette facilement. Mc est un ensemble d'anticorps *Ab, Mc= {Ab₁, Ab₂ .... Abₙ}* après la phase d'apprentissage servant à classer les nouvelles instances entrées dans le système (prédire les *Ag.c* des nouveaux *Ag*).

Avant de poursuivre la présentation des méthodes immunologiques d'apprentissage automatique, une petite présentation des théories principales dans lesquelles les AIS trouvent leur inspiration s'impose. Les idées essentielles de la majorité des méthodes immunitaires automatiques sont tirées de la théorie de sélection clonale, ainsi que la théorie du réseau immunitaire, la sélection négative et d'un degré moindre la théorie du danger [CAS 02].

1.4.2.1 La théorie de la sélection clonale

Proposée par Burnet [BUR 57] inspirée grandement de la théorie de l'évolution et la sélection naturelle darwinienne (*Figure 1*.13). Lorsqu'un lymphocyte (B-cell ou T-cell) se lie à l'antigène qui a causé la réponse immunitaire, les cellules prolifèrent en se clonant à l'ordre de plusieurs milliers de copies d'elles-mêmes et se divisent en deux types de cellules (plasma et cellules mémoire) [BRO 11]. Les cellules plasmatiques ont une courte durée de vie et produisent de grandes quantités d'anticorps, tandis que les cellules mémoire vivent pendant une période prolongée dans l'hôte en anticipant la reconnaissance future du même déterminant (antigène). La caractéristique importante de la théorie est que lorsqu'une cellule est sélectionnée et clonée, elle est soumise à de petites mutations (appelées Hypermutation somatique) afin de renforcer le lien avec l'antigène [DHA 96-CUT 02]. Lorsque l'antigène infecte le corps une seconde fois, ces cellules sont capables de produire des anticorps de haute affinité, ce qui va conduire à l'élimination de la maladie plus rapidement.

De manière générale, la méthode implique la sélection d'anticorps (solutions candidates) de la cellule mémoire prédéfinie (création aléatoire ou tirés de l'ensemble d'antigène), cette sélection est basée sur l'affinité (distance entre vecteurs) avec l'antigène. Les anticorps sélectionnés sont soumis à un clonage proportionnel à leur affinité. Les clones produits subissent une mutation inversement proportionnelle à leuraffinité, ce processus continue jusqu'à atteindre une condition donnée, ressemblant ainsi à des algorithmes évolutionnaires. Enfin, la technique choisit le clone avec la meilleure affinité comme cellule candidate à introduire dans la cellule mémoire [TIM 02-DAS 09-ZEK 15].

La sélection clonale est la théorie la plus populaire dans le domaine des systèmes immunitaires artificiels, donnant naissance à plusieurs méthodes de Data Mining [BRO 11]. Dans leur rapport technique, De Castro et Zuben [DEC 99] ont proposé l'algorithme de

sélection clonale (CSA), première application informatique de la théorie à des fins d'optimisation et de reconnaissance de formes. Plus tard l'algorithme fut renommé CLONALG (CLONal selection ALGorithm)[DEC 02-ZEK 15].

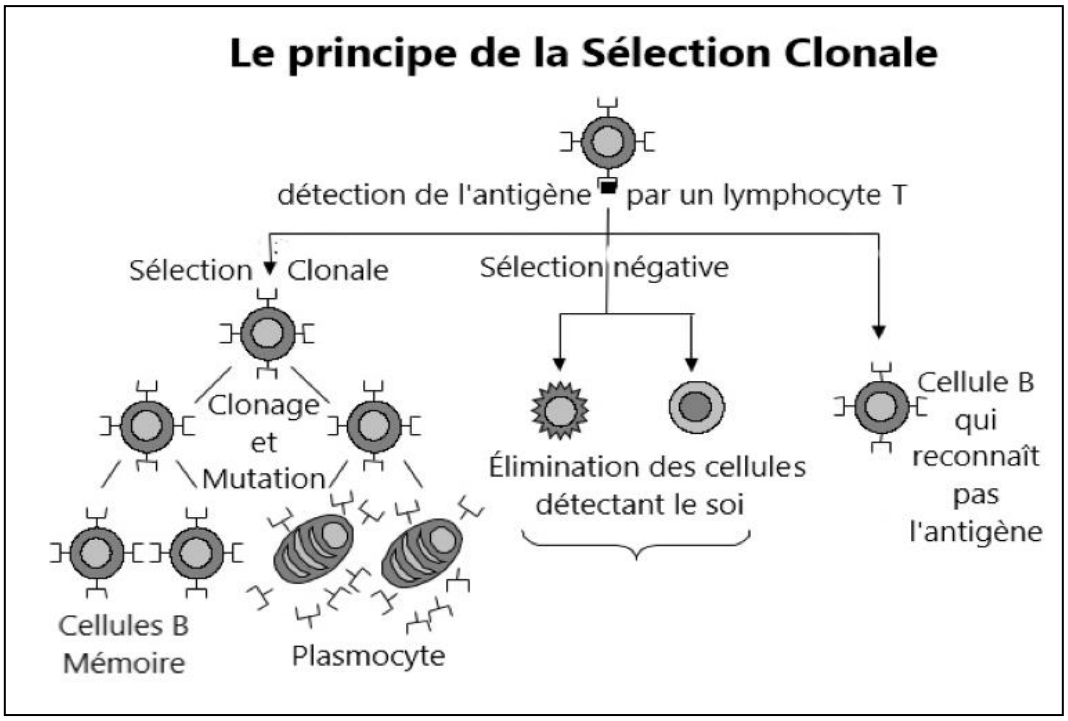


*Figure 1.*12-Le principe de la Sélection Clonale [4]

Watkins et al [WAT 03] ont proposés une version parallèle de l'algorithme d'apprentissage automatique. White et Garret [WHI 03] ont également étudié CLONALG pour la reconnaissance des formes et ont généralisé l'approche pour des tâches de classification, notamment binaires, en le renommant Clonal Classification (CLONCLAS). Garrett [GAR 04] a proposé Adaptive Clonal Selection (ACS) pour résoudre les problèmes inhérents de CLONALG telle que la représentation des fonctions continues lors d'une optimisation. Cutello et Nicosie [CUT 05] proposent une étude approfondie du principe de la sélection clonale et des algorithmes qui s'en inspirent. Quant à Brownlee [BRO 05b], en plus de fournir une revue des algorithmes de sélection clonale, il a proposé une nouvelle méthode d'apprentissage automatique intitulée (Clonal Selection Classification Algorithm – CSCA). S'inspirant grandement de la théorie clonale, Watkins [WAT 01] a proposé son interprétation d'une méthode d'apprentissage automatique immunologique AIRS (Artificial Immune Recognition System) avec de multiples versions [WAT 02-WAT 04].

1.4.2.2 La théorie de la sélection négative :

Cette théorie se trouve au cœur du fonctionnement du système immunitaire, principalement quand on parle de la discrimination du soi et non soi [FOR 94-BRO 11-ZEK 15]. La sélection négative (*Figure 1.*13) est une étape importante pour la maturation des cellules blanches T et B, afin de prévenir les attaques du système immunitaire de son hôte. Les maladies auto-

immunes sont écartées par ce processus en éliminant les cellules lymphatiques capable de reconnaitre l'épitope du soi (la marque du tissu de l'hôte ou simplement son identité), en ne gardant ainsi que les cellules capables de l'identification des agents étranges et pathogènes. Ce processus naturel est observé dans le thymus lors de la maturation des T-cells (T pour thymus)[FOR 94-BRO 11].

La première tentative pour l'adaptation de la théorie fut par Forrest et al [FOR 94], le but de Negative Selection algorithm (NSA) est de détecter les changements subis dans un fichier. C'est en fait le principal objectif des méthodes inspirées par la sélection négative, sans oublier la détection d'intrusion ou d'anomalie. La classification ne représente pas un objectif principal pour ces méthodes malgré l'existence de versions d'algorithmes implémentées à cet effet. .

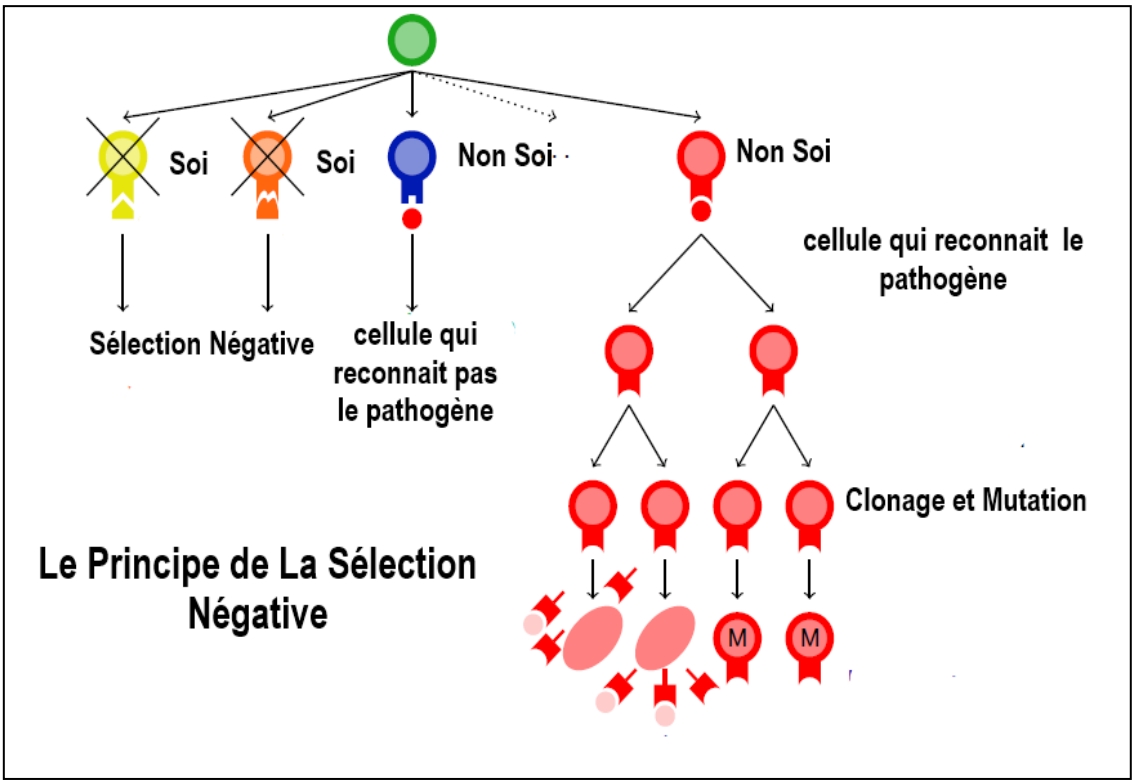


*Figure 1.*13-Le Principe de la Sélection Négative [JAN 12]

Le concept de l'algorithme est de générer aléatoirement plusieurs entités de détection, puis les séparer en deux parties, le groupe self et les instances non-self, ces parties sont le fruit de la sélection négative, la méthode se débarrasse du self en utilisant la base de données comme source d'information à propos du soi. Ainsi le groupe non-normal est produit afin de classer les anomalies, s'il y a une correspondance entre les instances du non soi et le nouveau entrant, généralement en utilisant la Distance de Hamming[5], ce dernier va être classé comme étant un intrus et sera bloqué par le système [FOR 94-BRO 11].

Gonzàlez et al. [GON 03] ont introduit un algorithme de la sélection négative à valeurs réelles (Real Valued RNSA) pour mieux maximiser la convergence de l'algorithme. Dans leur

[5] https://en.wikipedia.org/wiki/Hamming_distance

sillage, Ji et Dasgupata [JI 04] ont proposé une version améliorée de NSA, qu'ils ont appelée Augmented NSA, le principal changement est la capacité de création des détecteurs (T-cell) variable (vecteur du paratope variable) afin de s'accommoder avec les changements qui peuvent altérer l'espace du problème. Ji [JI 05] a modifié la méthode NSA, renommée Boundary-aware NSA, dans cette version au lieu que les instances d'apprentissage soient traité de façons individuelle l'algorithme procède par batch. Jinquan et al. [JIN 09] ont mis en œuvre ANSA (Adaptive NSA), qui permet de construire un modèle de détection robuste avec très peu d'information sur le self (le soi).

1.4.2.3 La théorie du réseau immunitaire (Immune Network) :

Introduite par Jerne [JER 74] au début des années 70, pour faire face à l'inactivité des cellules immunitaires réactives lorsqu'il n'y a pas d'agent pathogène auquel répondre comme proposé par le modèle de la sélection clonale. Ainsi la théorie suppose que, en absence d'agent pathogène, les anticorps et les cellules immunitaires se reconnaissent et se répondent mutuellement gardant le réseau en constante activité. [VER 89-NEU 92]. Les anticorps (à la fois flottants et liés) possèdent à la fois des paratopes et épitopes auxquels les récepteurs d'autres anticorps peuvent se lier. À la suite des interactions avec les récepteurs, le répertoire devient dynamique, où les récepteurs s'inhibent et s'excitent continuellement dans des réseaux idiotypiques. La théorie suggère que le processus de sélection clonale peut être déclenché par les cellules et molécules immunitaires, en plus du pathogène. Le processus de maturation (Mutation) s'applique à la fois aux récepteurs et épitopes qu'ils exposent [DEC 01-BRO 11].

Un des premiers algorithmes inspirés par la théorie des réseaux immunitaires artificiels fut proposé par De Castro et Zuben [DEC 01] appelé artificial immune NETwork (aiNET) pour l'analyse de données et l'optimisation. Le principe de la méthode est de préparer un répertoire de détecteurs pour résoudre un problème donné. Les cellules les plus performantes suppriment les cellules à faible affinité (similaires) dans le réseau, grâce à un processus interactif lié à l'exposition des pathogènes aux éléments du réseau (les anticorps), déclenchant ainsi le processus d'Hypermutation somatique propre à la théorie clonale [BRO 11].

Tous les modèles supposent une configuration initiale du réseau immunitaire, dans certains cas, la première configuration est produite aléatoirement, La plupart des applications des réseaux immunitaires artificiels commencent par un ensemble de données d'entrée qui correspond à un ensemble d'antigènes stimulant le réseau immunitaire, qui passe par un processus dynamique, jusqu'à ce qu'il atteigne la stabilité. Selon les cas, le résultat final du réseau n'est autre que la concentration de chaque type d'anticorps ou sa structure et, dans de rares cas, les deux [DEC 01-BRO 11-ZEK 15].

Même si la théorie du réseau immunitaire n'est pas aussi populaire en comparaison aux deux méthodes précédentes, elle fait l'objet de plusieurs implémentations informatiques surtout dans le domaine du clustering comme les travaux de Timmis et al [TIM 00] et leurs algorithmes AIN (Artificial Immune Network) et sa variante the Resource Limited AIN [12], ainsi que Artificial Immune Network (AINE) [KNI 01] par Knight and Timmis. Une

extension de la technique proposée dans [DEC 01] nommée opt-aiNet spécialement orientée pour l'optimisation multimodale par les mêmes auteurs dans [DEC 02 b]. Plus tôt Farmer et al [FAR 86] étaient les premiers à suggérer une utilisation plausible de la théorie dans le domaine du Machine Learning.

1.4.2.4 La théorie du danger

C'est la plus récente des théories pour l'acquisition d'une immunité, proposée par Matzinger en 1994 [MAT 94- MAT 02], et suggère que le système immunitaire ne réponde pas au non-soi mais au danger (*Figure 1.*14). Ce qui signifie que le système immunitaire n'attaque pas systématiquement tout ce qui est étranger à l'organisme, par exemple la nourriture. Le danger est mesuré en fonction de la gravité des dégâts sur les cellules, par l'envoi de signaux de détresse lors d'une mort inhabituelle. Quand une cellule envoie un signal de détresse, les antigènes proches de la zone de danger sont capturés par les cellules présentatrices d'antigènes (Macrophage) exhibant ainsi l'épitope aux lymphocytes. De plus, les cellules B possédant des récepteurs qui reconnaissent les antigènes de cette même zone, sont stimulées et s'engagent dans le processus de la sélection clonale. D'un autre côté, les cellules qui se trouvent en dehors de la zone de danger ou qui ne reconnaissent pas le photogène ne seront pas activées [DAS 09].

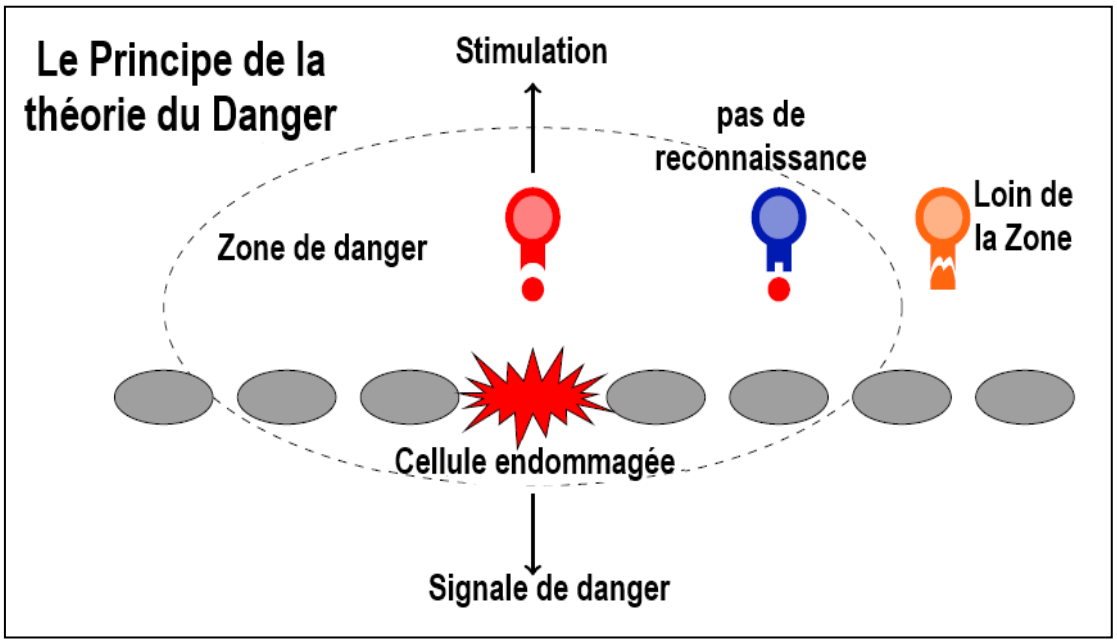


***Figure 1.*14-Illustration de la théorie du danger**

Le premier effort pour l'implémentation de cette théorie dans le domaine informatique et plus spécialement en Data Mining fut l'algorithme des Cellules dendritiques (Dendritic Cell Algorithm DC) par Greensmith [GRE 07], Il existe trois types de cellules dendritiques : immatures qui collectent des parties de l'antigène et les signaux, semi-matures qui sont des cellules immatures qui décident en interne que les signaux locaux sont sûrs en conduisant à une tolérance. En dernier les cellules matures qui décident en interne que les signaux représentent un danger et présentent l'antigène aux lymphocytes T, entrainant ainsi une réponse réactive [BRO 11].

L'objectif de l'algorithme est de préparer un ensemble de cellules dendritiques matures (prototypes) qui fournissent des informations sur la façon de classer les instances étudiées en normales ou anormales. Ceci est réalisé en trois étapes asynchrones : en premier la migration des cellules immatures suffisamment stimulées par le signal du danger, puis promouvoir ces cellules vers le statut semi-mature (sûr) ou mature (danger) en fonction de leur réponse accumulée généralement en utilisant un simple mécanisme de vote. Enfin, l'étiquetage des modèles observés comme sûr ou dangereux en fonction de la composition de leurs sous-populations de cellules [GRE 06-GRE 09].

Ces quatre théories représentent le cœur de toutes les méthodes inspirées des systèmes immunitaires avec un certain degré de réussite dans plusieurs domaines comme l'optimisation, la détection d'intrusion ou d'anomalie et dans l'apprentissage automatique. Presque ou toutes les méthodes AIS pour le Machine Learning s'appuient sur la théorie de la sélection clonale, comme CLONALG [DEC 02], AIRS (Artificial Immune Recognition System) [WAT 01-WAT 02], ces algorithmes et leurs multiple versions vont être explorées plus en détails dans la section suivante.

1.5 Systèmes Immunitaires Artificiels pour l'apprentissage automatique

Cette section va être consacrée aux méthodes d'apprentissages automatiques issues des systèmes immunitaires, plus précisément des algorithmes d'apprentissage supervisés utilisés dans les multiples travaux présentés dans cette thèse aux chapitres 3 et 4.

1.5.1 Système immunitaire artificiel pour la classification (AIRS) :

C'est l'un des premiers algorithmes d'apprentissage supervisé inspiré par l'immunologie, s'appuyant sur la théorie de la sélection clonale et négative pour l'acquisition de protection immunitaire, ainsi qu'une petite partie du concept des réseaux immunitaires qui se trouve dans les travaux de Timmis et al [TIM 00]. AIRS ou Artificial Immune Recognition System fut introduit par Watkins en 2001 dans sa thèse de master [WAT 01]. Il existe 3 variantes de cet algorithme dans la littérature qui sont AIRS 1 et 2 ainsi que la version AIRS2 Parallèle [WAT 01-WAT 02-WAT 04].

Selon Brownlee [BRO 05], l'algorithme est très complexe d'un point de vue implémentation, notre description va s'appuie grandement sur son rapport technique [BRO 05]. Le but de l'algorithme en soi n'est que de construire une cellule mémoire (*Mc)* qui va représenter la solution à notre problème, ici, un problème d'apprentissage. La *Figure 1*.15 décrit schématiquement le processus d'apprentissage AIRS et la *Table 1*.1 expose le rôle des paramètres utilisateur aidant à mieux adapter la méthode à son environnement.

Le déroulement de l'apprentissage d'AIRS est divisé en quatre étapes, comme suit [WAT 02-BRO 05] :

Étape 1 : L'initialisation :

Dès le départ les données d'apprentissage (antigènes *ag*) sont normalisées, de sorte que tous les éléments du vecteur de caractéristiques aient une valeur comprise dans [0,1].

Après, la cellule mémoire (*Mc*) est amorcée (seed). L'amorçage de $Mc=\{ab_1, ab_2, ...ab_n\}$ consiste à sélectionner aléatoirement un certain nombre d'antigènes pour devenir des cellules candidates (anticorps ab) à la résolution du problème. Le nombre de cellules candidates est un paramètre utilisateur, s'il est égal à 0, la méthode va s'occuper elle-même de la création des Ab.

Enfin, un seuil d'affinité (affinity threshold AT) est calculé selon l'équation suivante :

$$\boldsymbol{AT = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} affinité(agi,agj)}{\frac{n(n-1)}{2}}} \quad \textbf{(1.10)}$$

Où n est le nombre d'antigène dans la base d'apprentissage et affinité est **la distance euclidienne** entre deux antigènes. Affinity threshold AT va être utilisé plus tard pour déterminer si les nouvelles cellules *ab* candidates générées lors des étapes 2 et 3 peuvent remplacer les cellules mémoire existantes dans le classificateur.

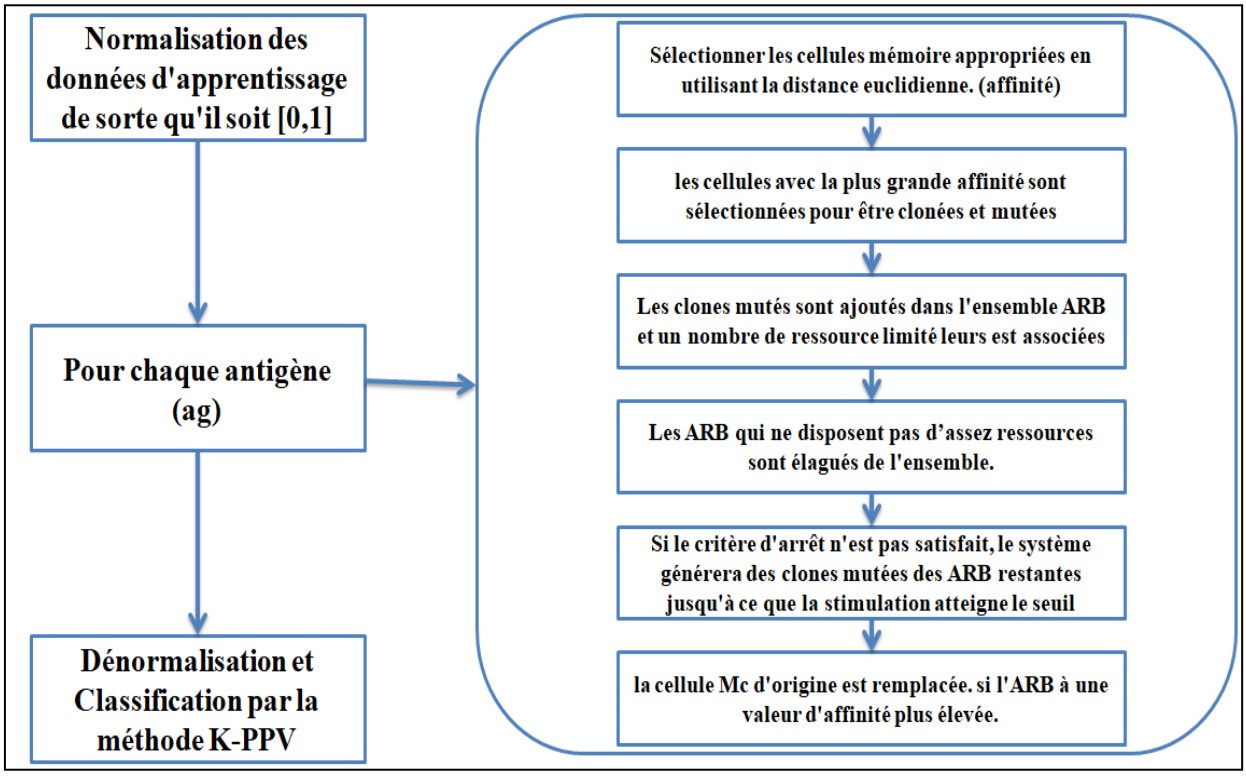


*Figure 1.*15- Représentation schématique le l'algorithme AIRS

Étape 2 : Sélectionner la meilleure cellule candidate :

Cette étape et la suivante seront exécutées pour chaque antigène dans la base d'apprentissage, chaque instance *(ag)* de la base est exposée à l'ensemble des cellules mémoire Mc *(ab)* qu'une seule fois (single-shot). La meilleure cellule est choisie selon sa valeur de stimulation *(S)* calculée de la manière suivante :

$$\boldsymbol{S = 1 - affinité(ag, ab)} \quad \textbf{(1.11)}$$

Après la sélection de la meilleure cellule candidate *best (ab)*, le processus de l'Hypermutation est déclenché, où le système crée plusieurs clones et il les fait muter afin de les préparer à prochaine étape, qui va raffiner cet ensemble de clones et *best (ab)*pour sélectionner la meilleure cellule possible afin de l'ajouter dans la cellule mémoire représentant la solution au problème. Le nombre de clones créés est estimé selon équation (1.12).

$$\boldsymbol{NumClone = S * clonalRate * hypermutationRate} \textbf{ (1.12)}$$

Où, S est la valeur de stimulation de *best (ab)* avec l'antigène, clonalRate et HypermutationRate sont des paramètres utilisateur. Ces clones et *best (ab)* vont être introduit dans l'ensemble *ARB*, La cellule *ARB* va jouer le rôle d'un espace de travail compétitif afin de sélectionnerle meilleur clone pour être une cellule candidate à ajouter dans l'ensemble *Mc*.

Étape 3 : Compétition pour les ressources limitées :

Maintenant que l'ensemble *ARB* est construit, l'étape compétitive commence afin de ne promouvoir que les cellules avec une grande valeur de stimulation *(S)* donc d'affinité. En premier lieu, l'antigène est réexposé aux cellules *ARB* et une valeur de stimulation est calculée comme dans l'étape précédente, ensuite un nombre de clones mutés sont générés pour tout l'ensemble des cellules *ARB* selon l'équation suivante :

$$\boldsymbol{NumClone = S * clonalRate} \textbf{ (1.13)}$$

Enfin, à chaque clone est attribué un certain nombre de ressource (R) selon son affinité avec l'antigène, le nombre de R pour chaque cellule ARB est le résultat de ce qui suit :

$$\boldsymbol{R=S*ClonalRate} \textbf{ (1.14)}$$

Ces ressources sont limitées selon un paramètre utilisateur (*total resources)*, qui va représenter le nombre maximum de ressources à allouer dans *ARB.* Par la suite, le système va trier en ordre décroissant les *ARB* selon leur valeur de *R*. Après, l'algorithme va élaguer les *ARB* en commencement par le bas (un nombre faible de ressources veut dire une affinité faible) jusqu'à ce que la somme des ressources allouées soit inférieure ou égale à *total resources.*

L'étape 3 est répétée jusqu'à atteindre une condition d'arrêt spécifié par le superviseur, où la valeur de stimulation *(S)* des *ARB* est supérieure ou égale à stimulation threshold.

Étape 4 : Introduction dans la cellule mémoire :

Après la fin de l'étape de la compétition pour les ressources limitées, la meilleure cellule ARB est sélectionnée comme solution au problème et ajoutée dans l'ensemble *Mc*, l'introduction ne se fait que si la valeur *S* de la *ARB* est supérieure à celle de la meilleure cellule candidate *best (ab)* originale trouvée dans l'étape 2.

Avant de terminer, le système vérifie s'il est nécessaire d'éliminer la *best (ab)* après l'introduction de la *ARB* dans *Mc*. L'élagage de la *best (ab)* se fait *si:*

$$\boldsymbol{affinité(ARB, best\ (ab))} < AT * affinityThresholdScalar \textbf{ (1.15)}$$

Où, *AT* est calculée dans l'étape 1 avec l'équation (1.10), affinityThresholdScalar est un paramètre d'utilisateur.

Les étapes 2, 3 et 4 sont répétées pour chaque antigène dans la base de données, le résultat de ces étapes d'apprentissage est un ensemble de cellule mémoire *MC* qui va représenter la solution à notre problème ou le modèle d'apprentissage de l'algorithme.

Lorsque le processus d'apprentissage est terminé, MC devient le noyau du classificateur AIRS. Les vecteurs de données contenus dans les cellules peuvent être normalisés ou laissés tels quels pour le processus de classification. La classification se produit en utilisant l'approche du k-plus proches voisins (k-Nearest Neighbor), où les k meilleures cellules correspondantes au nouvel entrant (grâce à la distance euclidienne) sont choisies, et la classe de cet étranger est élue par vote majoritaire.

*Table 1.*1-Les paramètres utilisateur de l'algorithme AIRS

Paramètre	AIRS1	AIRS2	Description
affinityThresholdScalar	✓	✓	Utilisé pour déterminer un seuil afin de remplacer les cellules mémoire lors de la phase d'apprentissage. valeur idéale entre 0.1 et 0.3
arbInitialPoolSize	✓	✗	Utilisé pour sélectionner le nombre initial d'instances dans l'ensemble ARB à partir de l'ensemble d'apprentissage de façon aléatoire.
ClonalRate	✓	✓	Détermine le nombre de clones à créer lors des étapes 2 et 3 de l'algorithme, la valeur recommandée est environ de 10.
hypermutationRate	✓	✓	Utilisé conjointement avec *ClonalRate* lors de l'étape 2 pour déterminer le nombre de clones à créer pour l'anticorps avec la meilleure affinité. Le taux idéal est de 2.
Knn	✓	✓	Utilisé lors de la classification, *Knn* est le nombre de cellules mémoires entrant dans le vote majoritaire afin de déterminer la classe de l'instance inconnue avec la méthode de K-plus proches voisins
memInitialPoolSize	✓	✓	Représente le nombre d'instances à utiliser comme cellules mémoires tirées aléatoirement de l'ensemble d'apprentissage, généralement il faut ne pas dépasser 20.
mutationRate	✓	✗	Le taux de mutation subie par un clone

numInstances ***AffinityThreshold***	✓	✓	le nombre total d'instance pour calculer *AT* lors de l'étape 1, -1 veut dire tout la base de données
stimulationValue	✓	✓	Stipule la condition d'arrêt de l'étape 3, généralement réglé à un nombre très élevé afin de garantir que le clone sélectionné ait une grande affinité avec l'antigène pour être une cellule mémoire candidate. la valeur idéale est de 0.9
totalResources	✓	✓	le nombre de ressources allouées lors de l'étape 3 afin de raffiner l'ensemble ARB, ce nombre est toujours limité entre 150 et 300.

Comme cité précédemment, il y a trois implémentations de l'algorithme AIRS. Ces trois versions sont appelées AIRS1, AIRS2 et AIRS2 Parallèle, où la dernière ajoute le concept de parallélisme, grâce au multithreading, à la méthode AIRS2 [WAT 04]. Selon Watkins et Timmis [WAT 02] AIRS1 est une implémentation obsolète de la méthode, et ils recommandent l'utilisation de la seconde interprétation. Cependant, très peu de changements sont à noter entre les deux techniques, selon Brownlee [BRO 05] AIRS2 est plus simple à comprendre et moins complexe, vu le nombre réduit de paramètres utilisateur en comparaison à la première version. Là où l'ensemble ARB est temporaire et change à chaque introduction d'un antigène, il est permanent dans AIRS 1 entrant ainsi dans la compétition pour les ressources limitées dans ARB. De plus, AIRS 1 permet la mutation de la classe des clones créés lors des étapes 2 et 3 de l'algorithme, ce qui n'est pas le cas dans AIRS2.

Enfin, la mutation dans AIRS2 suit le concept de l'Hypermutation, où les mutations sont inversement propositionnelles à l'affinité du clone, plus l'affinité du clone avec l'antigène est grande, moins l'est le taux de mutation, alors que dans AIRS1 la mutation des clones est liée à un paramètre utilisateur (*mutationRate*).

1.5.2 Les algorithmes de la sélection clonale (CLONALG/CSCA) :

CLONALG (CLONal selection ALGorithm) est le premier algorithme d'apprentissage automatique qui adopte complètement le concept de la théorie clonale pour l'acquisition d'immunité. Cette méthode fut, en premier lieu, citée dans les travaux de De Castro et Zuben sous le nom de CSA [DEC 99], puis renommée de la sorte en 2002 [DEC 02]. Les éléments de la sélection clonale sur lesquels la technique se base sont résumés comme suit [DEC 02-BRO 05 a -CUT 05]:

- Maintenance d'une cellule mémoire spécifique

- Clonage et mutation seulement des cellules qui ont la plus grande affinité avec l'antigène
- Génération et maintenance continuelle des cellules de reconnaissance afin de garantir leurs diversités, sans oublier l'élimination des anticorps qui ne sont jamais stimulés.

Comme toute méthode originaire du domaine de l'immunologie, le but de l'algorithme est de construire une cellule mémoire $Mc=\{ab_1,ab_2, ...ab_n\}$ représentant la solution au problème étudié, qui est l'apprentissage dans notre cas. Pour garantir une mémoire robuste, l'algorithme s'appuie sur deux méthodes de recherche afin de sélection la meilleure cellule candidate ab, la première c'est Hypermutation vue précédemment, où la méthode crée plusieurs clones mutés des cellules ayant la meilleure affinité avec l'antigène, de plus, ces mutations devraient être inversement proportionnelles au taux de stimulation pour la recherche locale. Tandis que la recherche globale est garantie par l'introduction continuelle des cellules créées aléatoirement par l'algorithme, représentant ainsi sa seconde méthode de recherche afin de sortir du maximum local et trouver les meilleurs anticorps *(ab)* pour la construction de la cellule mémoire [BRO 05 a].

La phase d'apprentissage de l'algorithme CLONALG est décrite ci-dessous, les paramètres utilisateur sont expliqués dans la *Table 1.*2. De plus, une représentation schématique de CLONALG est affichée dans la *Figure 1.*16 [DEC 02-BRO 05 a-BRO 11] :

1. **L'initialisation** : Dans cette étape, l'algorithme procède à la création d'un nombre défini par l'utilisateur *M* de cellules mémoires, plus précisément d'anticorps *Ab* de façon aléatoire. Après l'algorithme divise cet ensemble en deux parties ***m*** et ***r*** où m+r=M, ***m*** est l'ensemble de cellule *Ab* représentant la solution au problème d'apprentissage. Tandis que ***r*** va évoquer l'ensemble des cellules restantes susceptible d'être changé à chaque entrée d'un antigène afin de garantir la diversité de la population M.
2. **la phase d'apprentissage et sélection des cellules candidates** : cette phase de l'algorithme est composée de plusieurs étapes, répété un nombre ***G*** (paramètre utilisateur) de fois pour tous les antigènes (instances) dans la base de données, donc pour chaque antigène *ag* faire :
    a. Exposer ag à toute la population M (m+r) d'anticorps Ab et calculer l'affinité entres les Ab et ag, où l'affinité (Ab,ag) n'est que la distance de Hamming (équation 1.16) entre leurs vecteur.

$$\boldsymbol{D = \sum_{i=1}^{L} S} \text{ où } \boldsymbol{S = \begin{cases} 1 \; si \; ab = ag \\ 0 \; si \; ab \neq ag \end{cases}} \textbf{(1.16)}$$

    Où « L » est le nombre des composantes du vecteur Ab et ag.

    b. Trier les ***Ab*** selon leurs affinités avec ***ag*** en ordre décroissent, puis Sélectionner les ***N*** meilleures cellules pour les étapes suivantes. ***N*** est défini par le superviseur.

c. Dans cette étape, les ***N*** cellules sélectionnées sont clonées proportionnellement à leurs affinités avec ag. le nombre de clone (***ab****) pour chaque Ab dans N est le résultat de l'équation suivante :

$$\boldsymbol{cloneN = \left\lfloor \frac{\beta * M}{i} + 0.5 \right\rfloor} \quad \textbf{(1.17)}$$

Où β est le facteur de clonage supposé par l'utilisateur, *i* est le rang de Ab dans N.

d. Après l'étape de clonage, ces clones (***ab****) subissent une mutation inversement proportionnelle à leur affinité. Plus l'affinité est grande, moins sont les changements, afin de mieux s'adapter à **ag**, puis ils sont exposés une nouvelle fois, antigène et leur affinité est calculée avec (1.16).

e. La suite du processus d'apprentissage, s'intéresse à trouver la meilleure cellule candidate parmi les clones mutés (**ab*)** afin de l'introduire dans ***m***, le ***ab**** remplacera la cellule originaire dans ***m*** *ssi* l'affinité de la cellule candidate est supérieure à l'original.

f. Avant de terminer, l'algorithme sélectionne les ***d*** anticorps avec la plus basse affinité dans l'ensemble ***r*** et les remplace par des nouvelles cellules générées aléatoirement ; où ***d*** est un paramètre utilisateur.

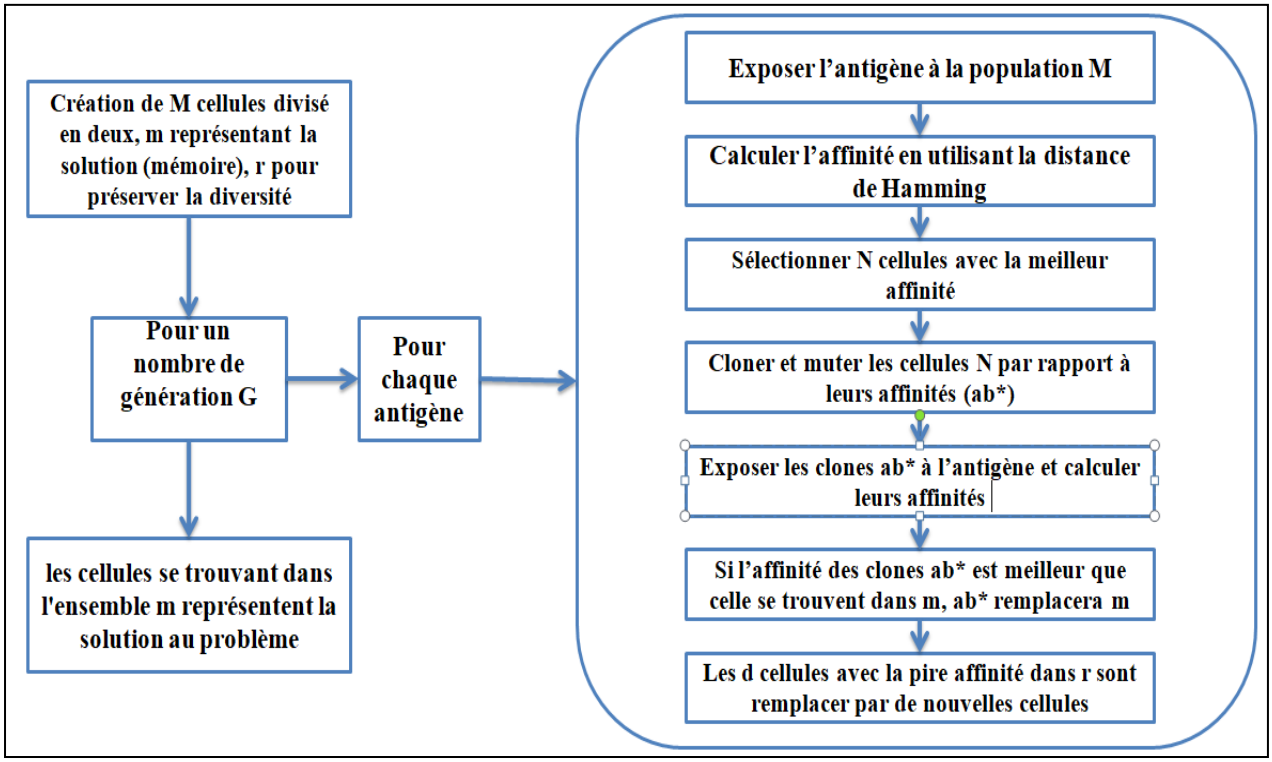


*Figure 1.*16- Représentation schématique le l'algorithme CLONALG

3. le dénouement : Après avoir répété les étapes a jusqu'à f pour chaque antigène dans la base de données d'apprentissage, et réitéré ce processus ***G*** fois. L'algorithme présent l'ensemble ***m*** comme solution au problème étudié, dans notre cas ***m*** représente l'ensemble d'anticorps utilisés pour classifier des instances inconnues.

Table 1.2- **Les paramètres utilisateur de l'algorithme CLONALG**

Paramètre	Description
M	Le nombre total de cellules à créer lors de la phase d'initialisation, par défaut 30 (antibodyPoolSize)
β	Le facteur de clonage, afin de déterminer le nombre de clones à créer pour un anticorps dans N. par défaut 0.1 (clonalFactor)
G	Le nombre de fois que tous les antigènes sont exposés à l'algorithme. Valeur recommandée est de 10 (numGenerations)
r	Le ratio de cellules allouées à l'ensemble des cellules restantes r. par défauts 0.1 (remainderPoolRatio)
N	Le nombre des cellules sélectionnée afin d'être clonées et mutées. Par défauts 20 (selectionPoolSize)
d	Le nombre des cellules à être remplacées avec la pire affinité dans l'ensemble r par de nouvelles cellules à chaque itération. Généralement c'est entre 5% à 8% (totalReplacement)

En premier lieu, CLONALG était une méthode orientée pour les problèmes d'optimisation plutôt que de classification. La meilleure ou les meilleures cellules avec la plus forte affinité parmi les cellules de la mémoire sont considérées comme solution au problème. Ce n'est qu'en 2003 avec les travaux de White and Garrett [WHI 03] que l'algorithme fut adapté à la classification sous le nom de CLONCLAS (CLONal selection algorithm for CLASsification)[6], où la classe prédite de l'instance inconnue est la classe de la cellule mémoire avec laquelle la cible a la plus grande affinité.

Dans sa démarche pour améliorer les performances lors des problèmes d'apprentissage de l'algorithme CLONALG et ses différentes variantes, Brownlee [BRO 05 a] a proposé une nouvelle méthode inspirée de la théorie de la sélection clonale spécifiquement pour le Machine Learning, cette méthode peut être vu comme une hybridation de deux concept AIS, l'algorithme CLONALG de base et la technique AIRS vu précédemment, cette technique est appelée CSCA (Clonal Selection Classification algorithm). L'innovation se trouve dans l'ajout d'une fonction *fitness* chère aux méthode évolutionnistes (méthodes inspirées du principe d'évolution darwinien comme les algorithmes génétiques), la fonction fitness est utilisée lors de l'étape de sélection, où l'algorithme ne s'appuie plus que sur l'affinité pour choisir les meilleures cellules candidates mais aussi sur la valeur de fitness, une cellules qui a une valeur fitness au-dessous un certain seuil ε sera éliminée par l'algorithme.

[6] Dans notre thèse nous allons continuer à l'appeler CLONALG car c'est le nom le plus commun même pour la classification pour faciliter la recherche.

Les détails de fonctionnement de l'algorithme CSCA sont présenté ci-dessous, de plus une représentation schématique de sa phase d'apprentissage est illustrée dans la *Figure 1*.17 et ses paramètre utilisateur son décrit dans la *Table 1*.3 [BRO 05 a- OLI 12] :

1. **Initialisation :** Comme tous algorithmes AIS, cette phase est importante, pour préparer la cellule mémoire qui servira de solution au problème d'apprentissage, l'algorithme va sélectionner ***M*** antigènes (ag) de la base d'apprentissage pour servir comme anticorps (ab) de la cellule mémoire. Là où la cellule mémoire est séparée en deux parties dans CLONALG ***M=r+m),*** CSCA déroge à cette règle, ***M*** représentera la solution à la fin de la phase d'apprentissage.

2. **Sélection des cellules candidates et élagage :** Comme l'algorithme original, cette phase va être répétée ***G*** (génération) fois, mais il y a une différence. Pour chaque ***G***, l'algorithme va exposer toute la population d'antigènes (instances de la base de données) à M (les anticorps) et non un par un.

    1. Lors de l'exposition de la population ***ag***, un score fitness est attribué à chaque ***ab*** dans ***M***, dans cet algorithme la valeur ***fitness*** va représenter le concept d'***affinité***. La valeur ***fitness* (*F*)** est calculé selon (1.18):

    $$\boldsymbol{F = \frac{B}{T}} \textbf{ (1.18)}$$

    Où ***B*** est le nombre d'antigènes correctement classifiés par ***ab*** dans ***M***, ou simplement le nombre de fois que la classe d'un ***ab*** a correspondu à la classe des antigènes constituant la base de données. Tandis que ***T*** c'est le contraire. Bien entendu ***B*** et ***T*** sont calculés que pour l'ab avec la plus grande affinité avec l'ag, estimé en utilisant la distance euclidienne.
    2. Dans cette partie, l'élagage des cellules inutiles commence en suivant les règles suivantes :
        1. Les anticorps avec ***B=0*** et ***T=0*** sont supprimés de M.
        2. Dans le cas où ***B=0*** et ***T> 0*** l'algorithme réajuste la classe d'ab est ***F*** est recalculé.
        3. Enfin, si ***F*** est inférieure à un seuil ***ε*** défini par le superviseur, ab sera aussi élagué.
    3. Après la phase d'élagage, l'algorithme procède au clonage et mutation des ab non élagués. le nombre de clones pour chaque ab est défini selon l'équation suivante :

    $$\boldsymbol{Nombre\ de\ Clone = \left(\frac{Fi}{\sum_{i=1}^{M} Fi}\right) * \partial * n} \textbf{ (1.19)}$$

    Où ***Fi*** est la fitness d'ab en question, ***M*** est le nombre total d'anticorps, de plus $\boldsymbol{\partial}$ est le facteur de clonage, ***n*** est le nombre d'antigènes. La mutation utilise aussi la valeur de ***F*** comme ratio au changement à apporter aux clones.
    4. Enfin, les clones mutés sont introduits dans l'ensemble M. De plus, les cellules éliminées lors de 2.1 sont remplacés par de nouveaux ab générées aléatoirement ou sélectionnées de l'ensemble d'apprentissage.

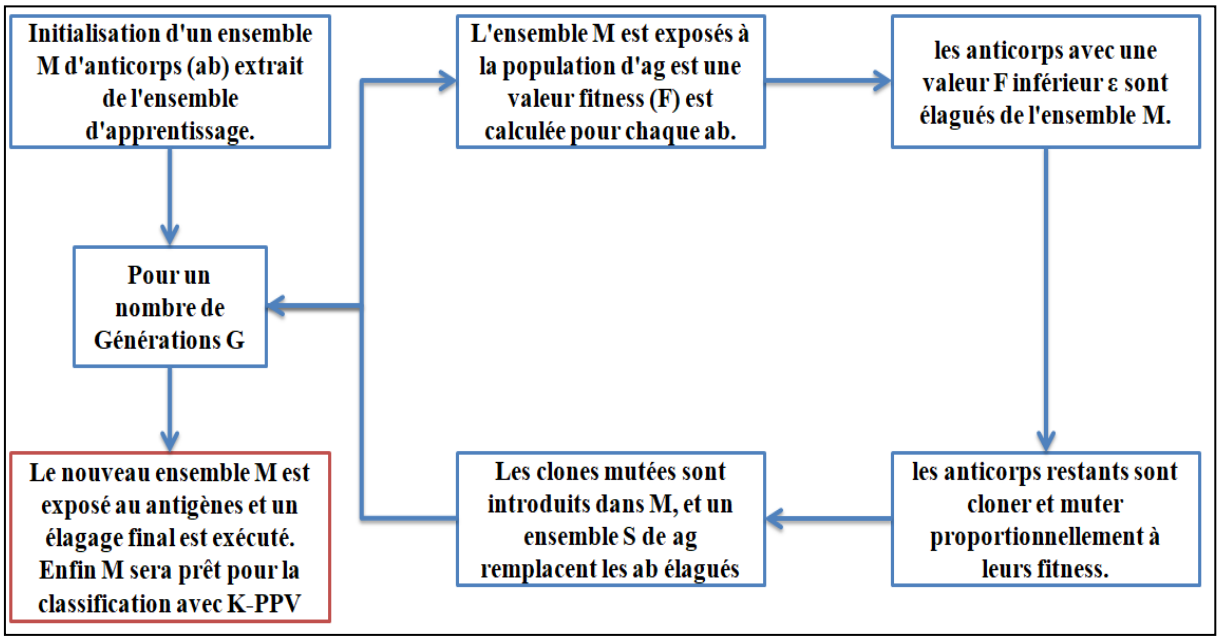


*Figure 1.*17- Représentation schématique le l'algorithme CSCA

3. L'élagage Final : l'ensemble M de cellules produites dans l'étape précédente est réexposé à la population d'antigènes, ensuite fitness est recalculée, ainsi donc un processus d'élagage final est enclenché suivant les même règles que ***2.1***. La cellule mémoire M finale sera considérée comme la solution au problème d'apprentissage.

*Table 1.*3- Les paramètres utilisateur de l'algorithme CSCA

Paramètre	Description
M	La population initiale de cellule mémoire de taille M. Généralement M est environ de 50 *(initialPopulationSize)*
∂	l'échelle de clonage des cellules candidate non élagués. La valeur par défaut est 1 (clonalScaleFactor)
ε	le seuil ε pour sélectionner les cellules à élaguer, généralement cette valeur minimale est environ 1 (minimumFitnessThreshold)
G	le nombre de générations à exécuter la phase 2 de l'algorithme. Selon Brownlee [BRO 05 a] il ne faut pas dépasser 10 générations pour éviter le **surapprentissage.**
Knn	Utilisé lors de la classification, *Knn* est le nombre de cellules mémoire entrant dans le vote majoritaire afin de déterminer la classe de l'instance inconnue avec la méthode de K-plus proches voisins

À l'instar de la méthode AIRS, la classification des instances inconnues dans CSCA est prédite en utilisant la cellule mémoire ***M*** et l'algorithme K-ppv (K-plus proche voisins).

1.5.3 Immunos-81 :

Considéré comme le premier algorithme d'apprentissage supervisé d'inspiration immunologique par le docteur en médecine Carter [CAR 00]. A cette époque tous les algorithmes AIS se tournaient vers le clustering (apprentissage non-supervisé) et l'optimisation des fonctions [DEC 99]. Carter [CAR 00] a déclaré que les précédentes méthodes AIS adhéraient beaucoup trop au concept biologique dont elles sont inspirées même s'il a admis que cela peut apporter quelques idées utiles sur le plan algorithmique, elles ne sont pas nécessairement essentielles d'un point de vue informatique [BRO 05b- KOP 06].

Selon Brownlee [BRO 05 b], en raison de la formation de Carter [CAR 00], Immunos-81 est présenté de façon atypique en comparaison aux travaux du même domaine, rendant la compréhension de la méthode très difficile pour un informaticien. Donc la suite de la section va grandement s'appuyer sur le travail de Brownlee [BRO 05 b] et ses efforts à expliquer les subtilités de l'algorithme, nous allons aussi utiliser notre **exemple** vu dans la *section 1.2* pour ajouter plus de clarté.

Avant de s'approfondir dans l'explication de la phase d'apprentissage de la méthode Immunos-81, quelques définitions des différents éléments Immuno-inspirés qui interviennent dans la technique s'imposent [BRO 05 b] :

- **La cellule T** : Ces cellules apprennent la structure primaire des antigènes et décident de la manière avec laquelle les nouvelles informations sont exposées au système. Il existe une cellule T pour chaque type d'antigène, et chacun des lymphocytes T possède un ou plusieurs clones (groupes) de cellules B. si nous prenons **l'Exemple de la section 1.2**, la structure apprise par la cellule T sera (Age ; poids ; Classe)
- **La cellule B** : Ces éléments apprennent les caractéristiques des antigènes (les valeurs des attributs appris par les cellules T). Les cellules B représentent une instance d'un type d'antigène, plus précisément une instance d'un groupe antigène (Clone). Il est mentionné par Carter [CAR 00] que les cellules B ne sont pas représentées explicitement par Immunos- 81 et sont plutôt incorporées dans une structure de groupe ou clone.
- **Antigène (AG) :** Représente l'instance à apprendre par l'algorithme.
- **Type d'antigène (TA)** : parfois confondue avec la classe de l'antigène par J.Carter [CAR 00], le type d'antigène désigne un domaine du problème identifié par un nom spécifique et une série d'attributs. C'est la structure primaire apprise par les cellules T (vecteur de caractéristiques). si nous prenons **l'Exemple** de la section 1.2, le type antigène sera comme suit : nom : judoka ; attributs : âge, poids, classe.
- **Groupe d'antigène (GA)** : un groupe antigène est un ensemble d'antigènes d'un type spécifique. Étant donné que la nature du problème abordé par Immunos-81 est

un problème de classification, un groupe antigène (clone) représente tous les antigènes avec une étiquette spécifique. Ce groupe est appelé clone de cellules B contrôlé par une seule cellule T. dans **l'Exemple** de la section 1.2 il y aura deux groupes, blessé ; non blessé.

- **Épitope/Paratope** : un Paratope est une partie du récepteur sur une molécule d'anticorps qui se lie à une partie correspondante de la molécule d'antigène appelé l'épitope.
- **Affinité (AF)**: Dans le système Immunos-81, l'affinité est une correspondance à une mesure de similarité, entre une cellule T et un antigène ou une cellule B et un antigène grâce à la distance euclidienne. Elle est utilisée lors de la classification des antigènes inconnus selon équation suivante :

$$\boldsymbol{AF = \sqrt{\sum_{i=1}^{N} di}} \quad \textbf{(1.20)}$$

Où *di* est la distance entre l'i^{ème} composant du vecteur caractéristique entre la cellule b et l'antigène. *N* est le nombre total d'attributs.

- **Concentration (Antigénicité)** : Un terme intéressant utilisé pour décrire le nombre d'antigènes (instances d'apprentissage) avec une étiquette de classe spécifique (groupe antigène). Cette concentration indique évidemment la quantité d'attention que chaque groupe antigène recevra, c'est ainsi la définition de la taille de clone du groupes antigène de cellules B.
- **Avidité (AV):** Avidité est le terme utilisé pour décrire la somme d'affinité de tous les clones d'un groupe de cellules B, cette valeur permet au clone de classifier l'antigène. le calcul d'AV se fait comme suit :

$$\boldsymbol{AV = \frac{CS}{CA}} \quad \textbf{(1.21)}$$

Où *CS* c'est le nombre total de clones d'un groupe de cellule B, *CA* c'est la somme des affinités *(AF)* de ce même groupe avec l'antigène. Cette équation permet de récompenser les clones avec la plus grande ***Concentration*** et la meilleure affinité afin d'étiqueter les instances inconnues lors de la classification.

- **Bibliothèque d'acide aminé (Amino Acid Library (aalib))** : Cette bibliothèque permet de centraliser le travail de l'algorithme. Elle aide les cellules T à connaitre les caractéristiques de chaque antigène, et facilite ainsi le travail de la sélection des cellules T et le clonage de cellule b. Aalib de **l'Exemple** de la section 1.2 se présente comme suit : nom : Age, type : numérique ; nom : poids, type : numérique ; nom classe, type : numérique.

Il y a deux versions implémentées de cette méthode par Brownlee [BRO 05 b] qui sont Immunos-1 et 2, car l'auteur n'a pas pu contacter Carter [CAR 00] afin d'avoir plus d'information sur les fonctionnalités de l'algorithme. La différence principale entre les deux versions est qu'Immunos-2 admet une phase de réduction des données. Immunos-2 construit

version condensée des clones (cellules B) de chaque groupe, qui n'est autre que la moyenne des valeurs de chaque attribut du vecteur de caractéristiques reconnu par la cellule T.

La phase d'apprentissage d'Immunos-81 est résumée dans la *Figure 1.18* et détaillée comme suit :

1. Initialisation : contrairement à tous les algorithmes AIS vus précédemment cette méthode n'a besoin d'aucune préparation, seul le fait que toute la base d'apprentissage soit présente.

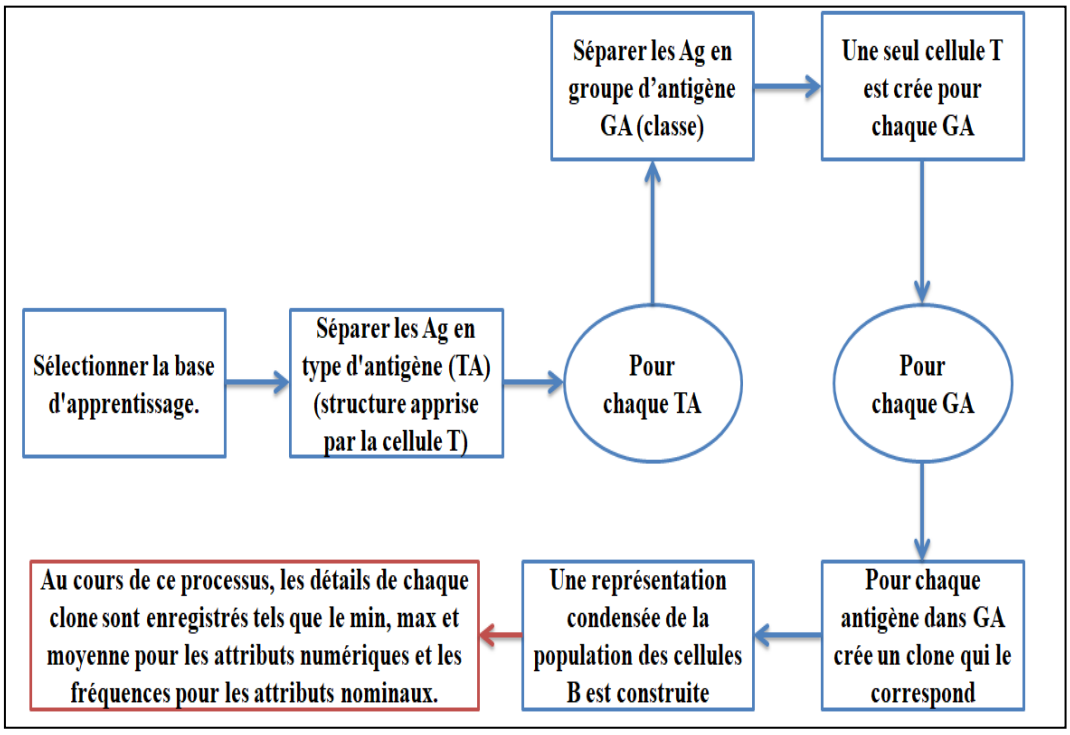


*Figure 1.*18- Représentation schématique le l'algorithme Immunos-81

2. Préparation des cellules T : L'une des forces d'Immunos-81 est que l'algorithme admet des données avec des vecteurs de caractéristique différents (type d'antigène), cette fonctionnalité est attribuée aux cellules T.

a. Grouper les antigènes par rapport à leur type d'antigène (TA).
b. Créer une cellule T pour représenter un TA.
c. Séparer les antigènes en groupes d'antigènes (GA), chaque GA représente une classe pour l'apprentissage automatique.

3. Préparation des clones (Cellule B) : les groupes GA sont donc séparés et la cellule T, à laquelle ils répondent spécifiquement, leur est attribuée. Donc pour chaque GA :

a. Pour chaque antigène créer une cellule B qui lui correspond.
b. Après avoir créé tous les clones qui correspondent au groupe d'antigène sélectionné, le système considère qu'il a appris les spécificités de ce GA. le nombre de clones construits va représenter sa Concentration (Antigénicité).

c. Avant de passer à l'étape suivante, le système ou la version 2 de la méthode construit une cellule condensée des clones du groupe en exécution, en calculant les moyennes des valeurs de chaque attribut du Paratope (vecteur de caractéristique), ces détails sont sauvegardés dans la bibliothèque Aalib.

4. Classification : le résultat de la phase d'apprentissage n'est autre que des ensembles de cellules B (clones) groupées par leurs classes et un type d'antigène (cellule T), lors de la classification, ces groupes s'affrontent afin de déterminer la classe de l'antigène (instance) inconnue. La *Figure 1.19* résume la phase de classification de l'algorithme Immunos-81.

Nous n'avons pas abordés les paramètres utilisateur des deux versions de l'algorithme Immunos-81 car ils n'existent pas, ce qui peut dissuader les chercheurs à utiliser la méthode car il y a aucun moyen d'ajuster le paramètre du système afin d'améliorer ses performances. Dans ce sens, Brownlee [BRO 05 b] a proposé une nouvelle version d'Immunos-81, qui n'est autre qu'une interprétation simpliste de l'algorithme CSCA [BRO 05 a] vu précédemment avec des concepts propres à Immunos-81. Cette hybridation appelée Immunos-99 [BRO 05 b], ajoute des principes de la sélection clonale à Immunos-81, comme le clonage et mutation des cellules B responsable de la classification des entités inconnues. Cette version est plus facile à comprendre que les autres, car nous avons déjà couvert tous les principaux axes du fonctionnement de la méthode.

Ci-dessous nous allons expliquer les étapes de la phase d'apprentissage d'Immunos-99 [BRO 05 b] :

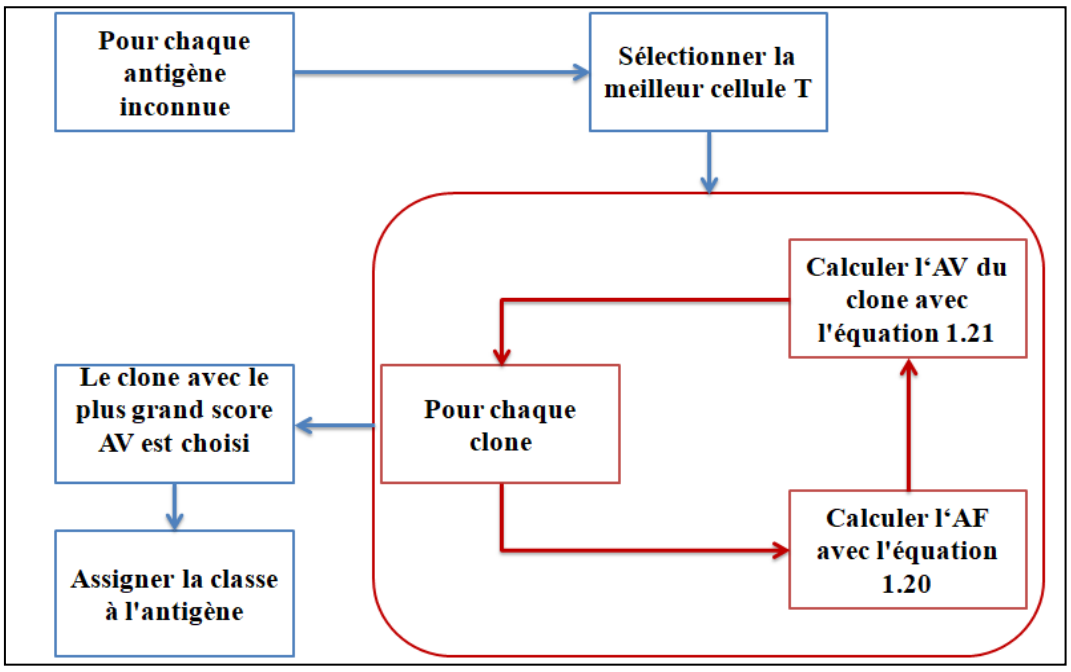


Figure 1.19- La phase de classification de l'algorithme Immunos-81

1. Initialisation : Au contraire des deux versions précédentes d'Immunos-81, cette méthode va subir une initialisation chère à presque toutes les méthodes immuno inspirés. Tout

d'abord l'algorithme va séparer l'ensemble d'apprentissage (les antigènes) en groupes, selon leurs classes (groupe d'antigène dans Immunos-81). Puis le système va choisir S antigènes (paramètre utilisateur) pour établir l'ensemble des cellules B qui va servir comme solution au problème d'apprentissage.

2. Préparation des Cellules B: Après avoir établi l'ensemble *S* de cellules B, le système crée aléatoirement de nouvelles cellules afin d'égaler le nombre d'antigènes présent dans la base d'apprentissage. Pour un nombre de générations *G* donné par le superviseur :

a) Exposer la population de cellules B à tous les antigènes peu importe leurs groupes. Puis calculer l'affinité des cellules B avec chaque antigène en utilisant l'équation (1.20).
b) Ordonner les cellules B par rapport à leurs affinités avec un antigène. L'ordre est croissant car selon l'équation 1.20 plus la valeur d'affinité est basse plus la cellule est meilleure.
c) Comme dans la méthode CSCA le système va faire intervenir le concept de fitness (*F*). La valeur *F* est établie selon l'équation suivante :

$$F=\frac{Cr}{Ir} \quad (1.22)$$

Où *Cr* est égale à la somme des rangs de la cellule B en question lors de son exposition à chaque antigène et que la classe de la cellule correspond à la classe de l'antigène dans l'ensemble d'apprentissage. Tandis que *Ir* c'est pour le contraire.
d) Élaguer toutes les cellules qui ont une valeur de fitness inférieure au seuil défini par l'utilisateur **ε.**
e) Cloner puis muter les cellules restantes, le nombre de clones pour chaque cellule est défini par l'équation 1.19. Par contre, il faut utiliser le ratio du rang de la cellule mère et pas la valeur fitness elle-même comme dans CSCA. Plus la cellule B est bien classée, plus la proportion de mutation des clones diminuent.
f) Après avoir ajouté les clones à l'ensemble de cellule B, de nouvelles cellules sont introduites afin de remplacer les cellules élaguées dans l'étape précédente. Ces nouvelles cellules B sont en fait des antigènes tirés aléatoirement de l'ensemble d'apprentissage dans l'intention d'ajouter plus de diversité et sortir des maximums locaux.

3. Élagage final : Avant de présenter les cellules B candidates pour l'étape de classification, l'ensemble de ces cellules subit un nouvel élagage comme dans CSCA, où elles sont exposées, encore une fois à l'ensemble des antigènes. Cette fois, l'algorithme n'utilise pas le rang des cellules puisque le score ne va être calculé que pour la meilleure cellule, exactement comme dans l'algorithme CSCA. Enfin, l'ensemble final de B-cell est préparé pour la classification.

4. Classification : Contrairement à la méthode CSCA qui utilise l'algorithme K-plus proches voisins afin de déterminer la classe de l'antigène inconnue. Immunos-99 exécute le même principe de classification que la méthode originale Immunos-81 vu dans la *Figure 1*.19, sauf qu'il n'existe pas de cellule T dans cette version hybride.

Enfin, tous les algorithmes et leurs multiples versions discutées dans cette section sont implémentés en langage Java, puis insérés dans l'outil WEKA[7] (Waikato environment for knowledge analysis- Environnement Waikato pour l'analyse de connaissances) [HOL 94-HAL 09] par Brownlee[8] [BRO 05-BRO 05 a-BRO 05 b].

1.6 Conclusion

Dans ce chapitre, nous avons abordé le domaine de Data Mining (Fouille de données) en nous concentrant sur un de ses champs qui est l'apprentissage automatique (Machine Learning ou ML). Sommairement, ML est l'étude des méthodes qui octroient aux machines la capacité d'apprendre et d'inférer des connaissances à partir d'un un ensemble de données, généralement sous forme de base de données. Dans le cadre de cette thèse, nous nous sommes intéressés à des approches bio-inspirées d'apprentissage et de classification, les réseaux de neurones artificiels (ANN) et les systèmes immunitaires artificiels (AIS). Les ANN trouvent leur inspiration dans les éléments biologiques qui permettent à l'être humain d'apprendre et d'acquérir de l'expérience pour évoluer dans son environnement, ces petits éléments sont les neurones d'où le nom de cet algorithme.

Les AIS trouvent leur origine dans le système permettant aux humains de se protéger des maladies, virus, bactéries ou tout autre substance représentant un danger pour le fonctionnement des éléments vitaux du corps de l'hôte.

Comme nous nous intéressons particulièrement aux approches immunologiques, nous y avons consacré la plus grande partie de ce chapitre. nous nous sommes Après une petite présentation de ces méthodes, nous nous sommes focalisés sur l'explication des différents algorithmes d'apprentissage supervisés AIS et leurs multiples versions. Nous pouvons les regrouper en 3 catégories qui sont :

- Immunos-81 (Immunos-1, Immunos-2, Immunos-99) [CAR 00-BRO 05 b],
- Système immunitaire artificiel pour la classification (AIRS1-AIRS2-AIRS2 parallèle) [WAT 01-WAT 02 -WAT 03-WAT 04]
- Sélection clonale (CLONALG,CSCA) [DEC 02-BRO 05 a]

[7] https://www.cs.waikato.ac.nz/ml/weka/ https://sourceforge.net/projects/weka/

[8] https://github.com/fracpete/wekaclassalgos

Chapitre 2 : Prédiction de défauts logiciels (Software Fault Prediction)

La prédiction de défauts logiciels représente l'un des axes de recherche les plus actifs quand il s'agit de l'utilisation des méthodes de fouille de données en ingénierie de logiciels. Nous nous intéressons particulièrement à cet axe en raison de son apport indéniable au développement de logiciels.

Ce chapitre représente principalement un état de l'art de la prédiction des défauts logiciels (Software Fault Prediction "SFP"), contenant ainsi, les concepts de base de cette discipline, tout en exposant les bienfaits de l'utilisation de SFP. Sans oublier les outils et ressources essentiels pour accomplir cette tâche.

2.1 Introduction

La naissance du génie logiciel (Software Engineering "SE") comme discipline à part entière remonte à la fin des années 60, débuts 70, en réponse à la crise dite "crise du logiciel" déclenchée par la conférence NATO à Garmisch, Allemagne en 1968 [NAT 68]. Le besoin grandissant de logiciels plus complexes est traduit par des produits qui ne répondent plus aux exigences des clients tout en dépassant les budgets et le temps alloués aux projets [KAN 03]. Ainsi, la nécessité d'avoir une touche d'ingénierie dans le développement des logiciels s'est fait sentir, afin de mieux planifier et contrôler ce processus et arriver à satisfaire les clients tout en restant dans le budget et le temps attribué [WIR 08]. Ce processus s'appelle le cycle de vie du logiciels ou Cycle de développement (Systems development life cycle "SDLC").

Le plus simple des modèles de développement est le processus en cascade (the Waterfall processus) (*Figure* 2.1) introduit par Royce [ROY 70]. Ce processus oblige l'équipe de développement de préciser ce que le produit doit faire grâce à une bonne définition des exigences logicielles avant de le développer concrètement. Le modèle en cascade sépare les grandes lignes du SDLC en sous catégories (design, codage, test et maintenance) où le passage d'une étape à une autre exige une validation et vérification de la phase qui la précède, ce qui résulte par un produit répondant aux exigences du client tout en garantissant sa qualité [JAL 09].

À chaque étape du processus de développement cité précédemment, plusieurs documents et sous-produits sont délivrés, comme le code source, l'architecture du logiciel, le résultat des tests, les ressources humaines ou matérielles nécessaire pour arriver à terminer le projet ... Ces éléments représentent une mine d'information exploitable par des méthodes de Data Mining afin d'extraire des connaissances pouvant aider l'entité intéressée à mieux gérer ses futurs projets. Une des utilisations possibles de ces données est la prédiction du temps et budget à allouer à un certain logiciel [MOL 03-SHA 21], ou par exemple l'effort de maintenance [JOR

95-ZIG 18-ALS 20]. Plus encore la prédiction de défauts logiciels (ou, dans certains cas, la prédiction de la Qualité logicielle) qui pourrait aider l'équipe chargée du développement à ne viser que les composants défaillants du système pour réduire ainsi le temps et le budget impartis à une phase donnée [ZHA 03].

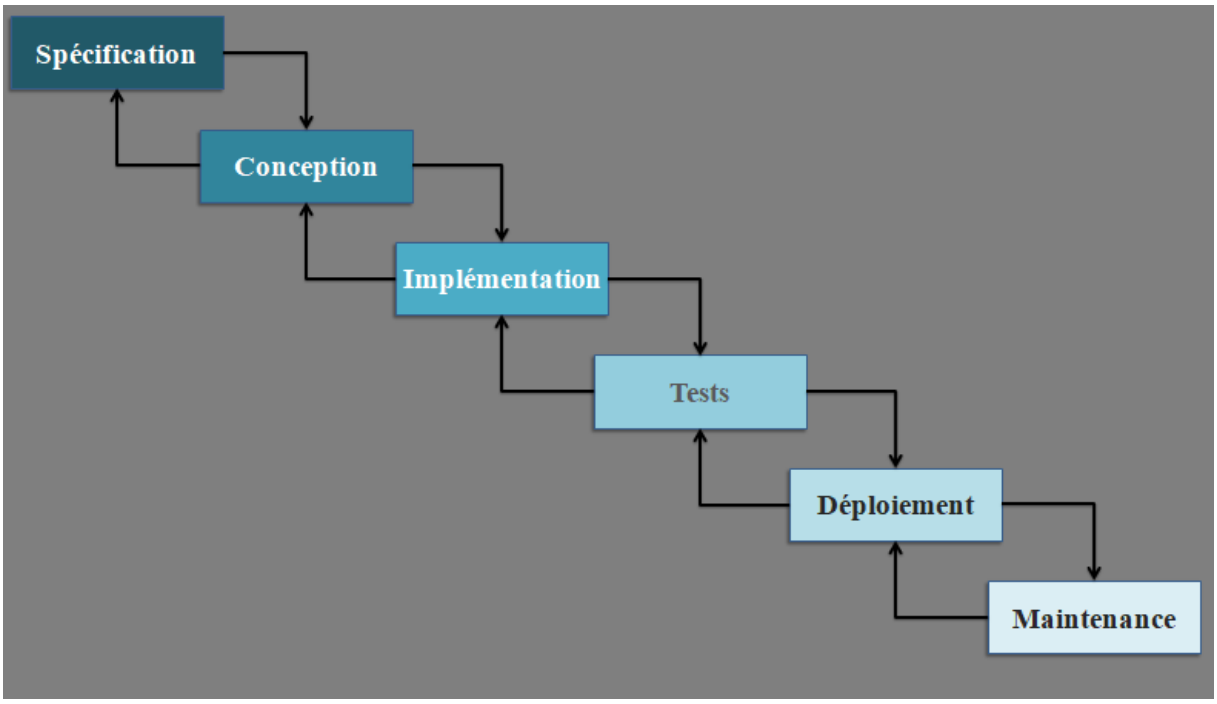


Figure 2. 1- le processus en cascade (Waterfall processus)

L'importance des logiciels à notre époque est indéniable. Presque tout ce que nous utilisons est un logiciel ou une entité contrôlée par un logiciel. Veiller à assurer le bon fonctionnent de ce produit est d'une importance capitale. Commençons par la première phase du développement qui est la spécification jusqu'au déploiement et la maintenance du système, le logiciel passe par plusieurs changements représentant un chalenge considérable pour les acteurs responsables de son implémentation, plus le système est large plus cette tâche devient difficile et laborieuse [MYE 08-BRO 87].

L'ingénierie logicielle (GL) est considérée comme étant la plus exigeante des disciplines d'ingénierie même si le public général n'est pas conscient de cet aspect puisqu'il n'a pas une vision concrète de sa taille et sa complexité. En effet, pour un édifice tel que l'Empire State Building[9] [BRO 87-HAR 12], là où l'absence d'une brique dans la tour ne conduit pas forcément à son effondrement, l'oubli ou le changement d'une simple ligne dans le code source d'un logiciel impactera forcément son fonctionnement [HAR 12].

Selon Jalote [JAL 09] le GL, s'intéresse à trois axes principaux qui sont la planification et le coût de développement, ainsi qu'à la qualité du produit. Donc le but est de produire un système de qualité tout en restant dans les temps avec le moindre coût possible. La qualité est

[9] L'Empire State Building est un gratte-ciel situé dans l'arrondissement de Manhattan, à New York.

un domaine du SE à part entière. Depuis la naissance du SE, la **Qualité logicielle (QL)** (Software Quality "SQ") a été et reste, à ce jour, le point focal de l'industrie logicielle [BOE 78-KAN 03-JAL 09-MIG 14-RIT 21]. La suite de ce chapitre va aborder les notions dequalité et de test logiciels et quelles sont les méthodes principales pour s'assurer que le système développé soit de qualité. Nous allons ensuite nous concentrer totalement sur les concepts et travaux liés à la prédiction de défauts logiciels.

2.2 Qualité logicielle (Software Quality) et Test logiciel (Software Testing) :

2.2.1 Qualité logicielle (Software Quality)

La qualité est un terme général qui prend plusieurs formes selon le point de vue et le domaine d'études [NAI 08], la qualité pourra signifier le luxe, coûteux ou élaboré ... Dans un sens la qualité est quelque chose que nous ne pouvons pas mesurer si nous prenons le point de vue populaire ou littéral. Par contre, le point de vue professionnel insiste sur le fait que la qualité est quelque chose de tangible, mesurable ainsi que contrôlable que chaque industrie doit atteindre [KAN 03]. Kitchenham et Pfleeger [KIT 96] définissent 5 niveaux de point de vue afin d'expliquer le terme qualité en ingénierie et plus précisément en GL :

- **Transcendantal** : l'idée est que la qualité est quelque chose facile à reconnaitre mais difficile à expliquer.
- **Utilisateur :** dans ce contexte, la qualité est mesurée par le degré de satisfaction de l'utilisateur.
- **L'industrie :** la qualité est définie par à quel point le produit est conforme à ses spécifications.
- **Produit :** dans le cas présent, elle est liée aux caractéristiques inhérentes du produit, de manière à ce que ses qualités intérieures aient un impact sur les qualités exposées.
- **Valeur :** c'est le nombre de clients qui sont intéressés par acheter le produit.

Selon le standard IEEE des terminologies en SE [IEE 90-IEE 06-SAM 10-MIG 14], la Qualité logicielle (QL) est définie comme un ensemble de règles et de principes à suivre au cours du développement d'une application afin de concevoir un logiciel répondant à ses spécifications ainsi qu'aux attentes des clients. La qualité d'un logiciel se reflète non seulement dans les processus de développement mais aussi dans la qualité des éléments qui le constituent, la documentation, la présence de test...

Mesurer la qualité d'un logiciel consiste alors à déterminer sa convenance par rapport aux objectifs de départ et aux standards de programmation. Il faut donc définir précisément ce que l'application doit faire et comment elle doit le faire, tant d'un point de vue fonctionnel que d'un point de vue technique. Une fois ces objectifs fixés, on peut alors appliquer un ensemble de règles et de mesures afin de calculer la différence entre les objectifs attendus et le produit final [KAN 03-MYE 08-NAI 08].

Depuis 1977, plusieurs modèles de SQ ont été proposés afin d'évaluer la qualité d'un logiciel développé. Un modèle de qualité n'est qu'un ensemble de caractéristiques et leurs relations servant de base aidant la spécification des exigences et l'évaluation de la qualité [ISO 01-DEI 09-AL 10-ISO 11-MIG 14]. L'un des premiers modèles de qualité fut proposé par McCall et al [MCC 77], il définit les relations entre les attributs qualité internes des applications et leurs caractéristiques externes. Ses caractéristiques reflètent les points de vue d'utilisateur ainsi que les développeurs. Dans ce modèle, trois caractéristiques principales sont considérées dans la définition et la sélection des attributs internes et de leurs relations qui sont : la Révision (Product Review) comme la maintenabilité, ainsi que la Transition (Product transition) comme la portabilité du système. Enfin l'Opération (Product Operation) comme la fiabilité. Elle est considérée comme base à tous les nouveaux modèles [DEI 09-AL 10-MIG 14], telle que Boehm [BOE 78], Dromey [DRO 95], FURPS[GRA 96] et enfin le standard ISO 9126 [ISO 01].

Le modèle ISO 9126 [ISO 01] (*Figure* 2.2) est le fruit d'une collaboration internationale d'expert sous la régie de l'Organisation internationale de normalisation (ISO). Ce modèle est composé de 6 critères principaux pour évaluer la qualité du système implémenté :

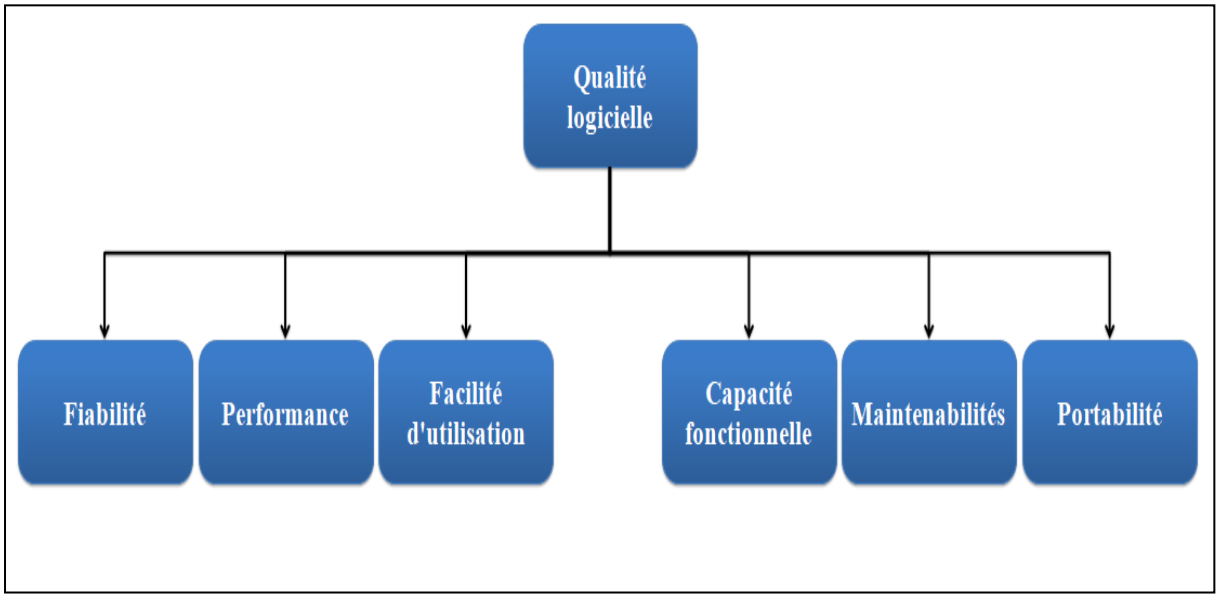


Figure 2. 2- Le modèle ISO 9126

- **Fiabilité (Reliability) :** La capacité du logiciel à maintenir son niveau de performance peu importe les conditions.
- **Performance (Efficiency)** : c'est la relation entre les performances du logiciel et les ressources matérielles ou logicielles disponible.
- **Facilité d'utilisation (Usability)** : c'est la facilité d'utilisation et d'apprentissage du logiciel.
- **Capacité fonctionnelle (Functionality) :** c'est la capacité du système à fournir les fonctionnalités exigées et nécessaires quand il est utilisé.

- **Maintenabilités (Maintainability) :** C'est l'aptitude à modifier, adapter et corriger le système.
- **Portabilité (Portability) :** c'est l'habileté du programme à fonctionner même en changeant l'environnement de travail, qu'il soit logiciel ou matériel.

Afin de s'assurer que le système réponde parfaitement aux attentes spécifiées, plusieurs méthodes sont disponibles sous le nom d'Assurance Qualité Logicielle (Software Quality Assurance "SQA") [TIA 05-SCH 07], la technique la plus utilisée parmi la sélection des méthodes SQA est celle des tests logiciels (Software Testing "ST").

2.2.2 Test logiciel (Software Testing) :

Lors du cycle de développement, des erreurs peuvent être introduite à n'importe quelle phase du projet, certaines peuvent être éliminées d'autres peuvent rester cachées. La meilleure façon de détecter ces erreurs est soit par une analyse statique par inspection du code,soit par une analyse dynamique en exécutant le programme lors des tests [JAL 09]. Ces deux méthodes se complètent afin de supprimer les défauts le plus tôt possible [MYE 08].

Les tests logiciels ont pour objectif de s'assurer que l'application fait ce qu'elle doit faire et rien d'inattendu [BEI 03-TIA 05-MYE 08-NAI 08]. Selon Myers et al [MYE 08] le but des tests est d'avoir une valeur ajoutée au programme comme hisser sa qualité ou fiabilité et non démontrer que le programme ne contient pas d'erreurs. Ainsi, il faut toujours assumer que le logiciel a des défauts et que les tests sont là pour les éliminer.

Économiquement parlant, il est presque impossible d'arriver à trouver toutes les erreurs que contient un programme [BEI 03-TIA 05-MYE 08-JAL 09-GAR 20], ce concept s'appelle le test complet (COMPLETE TESTING), car même pour un logiciel simple, trouver toutes les entrées possibles demande de très grandes ressources, qu'elles soient humaines ou matérielles, que les sociétés sont incapables de fournir [KIT 96-TIA 05-MYE 08]. Par conséquent, il faut, dès le départ établir des stratégies de tests, les deux stratégies les plus connues sont [NAI 08-MYE 08] :

- **Le test structurel (boite blanche/ white-box) :** le but est d'examiner le code source directement, il se concentre sur le flot de contrôle qui se réfère au passage d'une instruction à une autre de la partie testée. Ainsi qu'au flot de données, qui vérifie si les informations portées par les variables se propagent bien.
- **Le test fonctionnel (boite noir/black box)** : dans ce cas si le testeur n'as pas accès au code, il ne s'intéresse qu'aux entrées et aux résultats fournis par le programme telle une boite noire. Après, il vérifie si les sorties correspondent à celles prédites dès le départ lors de la construction des cas de test[10].

Il existe 4 niveaux de tests (*Figure* 2.3) à effectuer lors d'un cycle de vie du logiciel avant d'être déployé, ces tests sont les tests unitaires, les tests d'intégration, les tests système et enfin

[10] Des couples de entrées et sorties prédites afin de vérifier que le programme s'exécute comme spécifié.

les tests d'acceptation. À chaque niveau, différents acteurs sont responsables d'accomplir tel ou tel test dans le but de produire un produit le plus fiable et de la meilleure qualité possible.

La description du rôle de chaque niveau de test et les groupes impliqués sont présentés ci-dessous [BEI 03-TIA 05-MYE 08-JAL 09-GAR 20] :

- **Les tests unitaires (Unit Testing):** cette phase est effectuée par le programmeur lui-même, sur les plus simples unités du système telle qu'une méthode, classe ou composant.
- **Les tests d'intégration (Integration Testing) :** Après avoir accompli le test unitaire, le programmeur et l'ingénieur des tests d'intégration procèdent à assembler les entités du programme en de plus grande unités, et vérifient ainsi que le système reste stable avant de passer à la prochaine étape.
- **Les tests système (System Testing)**: Après avoir intégré toutes les unités du logiciel, les testeurs commencent à vérifier rigoureusement le système en employant une multitude de tests, tels que les tests de fonctionnalité, robustesse, sécurité, stabilité, fiabilité ... . Afin d'éliminer le plus de défauts possibles tout en s'assurant de ne pas dépasser les dates de livraison.

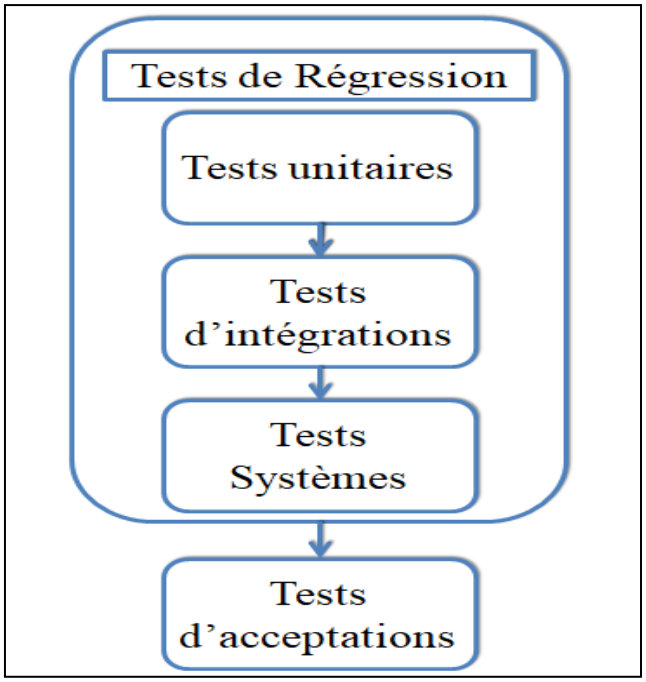


Figure **2. 3. Niveaux de Tests**

- **Les tests d'acceptation (Acceptation Testing):** dans cette phase, le logiciel est livré au client qui initie sa propre batterie de tests, de sorte à mesurer la qualité du programme. Ainsi le client vérifie si le système répond parfaitement à ses exigences.
- **Les Tests de régression (Regression Testing) :** Ils ne sont pas un niveau de test à proprement parler. Ils sont effectués à chaque modification, ajout ou suppression d'une ou plusieurs parties du système. L'objectif est de s'assurer que ces modifications n'ont pas introduit des nouveaux défauts aux parties non concernées

par l'erreur initiale qui a déclenché la modification. Les tests de régressions font partie intégrante des trois premiers niveaux de test.

L'utilité des tests est probante pour s'assurer de la qualité du logicielle. Les tests peuvent représenter jusqu'à 50 % du budget et temps alloué au projet, ce qui conduit parfois à complètement dépasser les coûts et délais fixés [SHE 95-MYE 08-NAI 08-HAS 21]. Des chercheurs étudient, depuis une trentaine d'années, l'intérêt d'utiliser des méthodes de data mining et, plus précisément, des algorithmes d'apprentissage automatique, pour réduire les coûts liés aux tests et d'augmenter, par la même occasion la qualité du système. Ce domaine de recherche s'appelle la **Prédiction de défauts logiciels (Software Fault Prediction « SFP »).**

2.3 Prédiction de défauts logiciels (Software Fault Prediction « SFP »)

2.3.1 Défauts (Defects), Erreurs (Errors), Défaillance (Failure) et Bug

Avant de définir la prédiction de défauts logiciels, il faut savoir ce qu'un défaut signifie d'un point de vu GL. Il est très difficile de trouver une référence francophone pour ce domaine de recherche. Ainsi nous allons faire de notre mieux pour expliquer les termes cités dans le titre de cette sous-section.

Un défaut logiciel pourrait survenir à n'importe quel moment du cycle de vie du logiciel, que ce soit lors de la phase d'analyse des besoins, conception, codage ou maintenance [CAT 09-KUM 18]. Un défaut peut prendre plusieurs aspects dépendant de son origine, comme un défaut de maintenabilité lors de la maintenance ou un bug lors de la phase de test [CAT 09-HAL 12]. Généralement, dans la littérature, les termes Défaut, Erreur, Défaillance et Bug sont utilisées de manière interchangeable [NAI 08]. Selon le standard IEEE 610-1990 [IEE 91] ces expressions sont définies comme suit [KUM 18] :

- **Erreurs (Errors)** : Ce terme désigne une action humaine conduisant à des résultats incorrects. Exemple : un programmeur qui oublié de mettre un point-virgule à la fin de l'instruction, ce qui résulte à un message d'erreur lors de la compilation.
- **Bug** : Dans le cas où le programmeur trouve des résultats inattendus lorsque le système est exécuté. Exemple : si le programme est sensé calculer l'addition de deux variables mais le résultat donné est leurs produit, la compilation cette erreur n'est pas détectable.
- **Défauts (Defects/ faults) :** Ce terme a le même sens que le précédent, par contre le défaut est souvent découvert par un modérateur, généralement le testeur. Cette formulation est la plus utilisé dans le monde du SFP.
- **Défaillance (Failure):** Si le logiciel ou une partie du système ne répondent plus aux exigences établies, ou n'offrent plus les fonctionnalités spécifiquement décrites, alors on dit que le programme est défaillant.

Un défaut logiciel représente un attribut de la qualité de ce dernier, qui explique l'incapacité du système à offrir les fonctionnalités désirées. Tandis que la défaillance logicielle est le symptôme causé par un ou plusieurs défauts [IEE91-NAI 08-KUM 18].

Dans la suite de cette thèse, les quatre termes établis précédemment sont utilisés de manière interchangeable. Par contre, ils se réfèrent tous à la définition du défaut (fault/defect).

2.3.2 Prédiction de défauts logiciels (Software Fault Prediction « SFP »)

D'une façon très simple nous pouvons définir la prédiction de défauts logiciels comme étant : la construction de modèle de prédiction à partir de données sous forme d'historique logiciels et des informations relatives aux défauts, venant d'un ou plusieurs anciens projets (*Figure* 2.4), en utilisant, le plus souvent, des méthodes d'apprentissage automatique. Le but est de déterminer si les entités (classes, méthodes, composants) du nouveau système sont sujettes ou non à contenir des défauts (faut-prone or not fault-prone) [SHE 95-LES 08-HAL 12-MAL 15-RAT 16-KUM 18-AME 19-ALG 20-PAN 21]. Ainsi, ce processus va aider les testeurs à ne viser que les modules sujets aux fautes, réduisant par la même occasion le temps et le budget alloués à cette phase du cycle de vie du logiciel, tout en augmentant la qualité du produit [CAT 09-CAT 11].

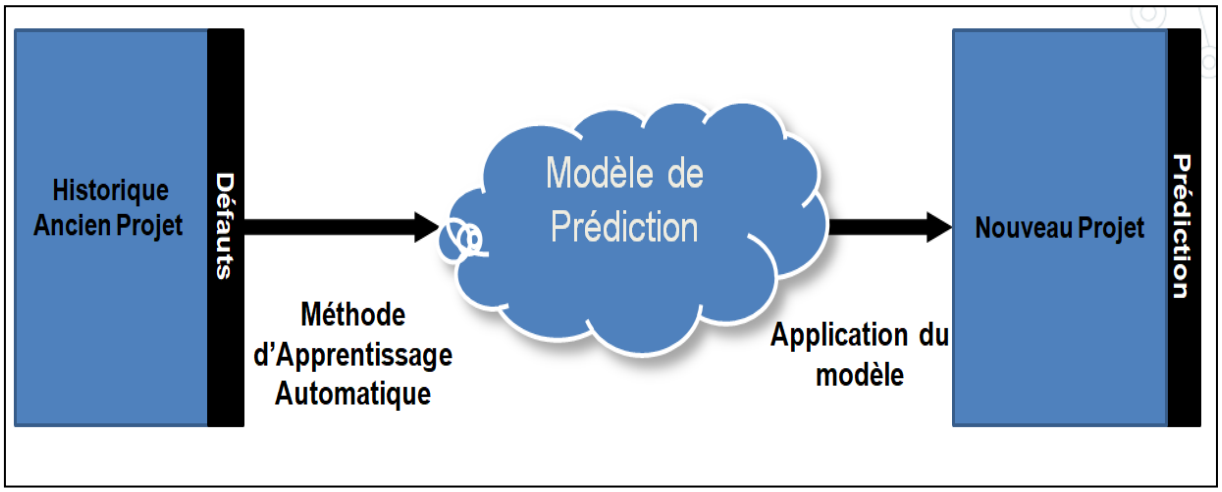


Figure 2. 4- Prédiction de défauts logiciels

Fréquemment, l'historique logiciel sur lequel sont bâtis les modèles SFP est composé de simples métriques logicielles, telles que le nombre de lignes de code (Lines of code "LOC") jusqu'à des mesures évaluant l'expérience des développeurs chargés du projet [MEN 10-CAT 12-RAD 13-MAD 15-CHO 18]. Plus encore, appliquer la prédiction de défaillances à des phases précoces du cycle de développement telle que la conception ou l'analyse des besoins logiciels, conduirait à améliorer l'architecteur du système et, éventuellement, réduire le nombre de bugs que pourrait contenir le produit final [GLA 00-JIA 08].

Dans leur tentative d'appliquer la prédiction de défauts logiciels dans le cycle de vie d'un projet réel, Monden et al [MON 13] ont déclarés que 20% de l'effort de test a été réduit sans impacter l'efficacité de la localisation des bugs, montrant ainsi l'utilité de l'utilisation de SFP dans le développement des logiciels. Nous pouvons résumer les bienfaits de l'utilisation de la

prédiction de faute dans un projet informatique dans les points suivants [CAT 11-ERT 15-KUM 18-RAT 19] :

- Le processus de test devient plus affiné, augmentant ainsi la qualité du système.
- La détection des modules qui nécessitent un *refactoring* lors de la phase de maintenance, est facilitée.
- L'application du procédé de prédiction lors de la phase de conception, permet de trouver une meilleure architecture.
- La prédiction de défauts du logiciel offre une stabilité aux systèmes logiciels.
- La prédiction de défauts du logiciel peut réduire le temps et les efforts consacrés au processus de révision du code.

Les travaux de recherche dans le domaine de prédiction de défauts logiciels peuvent être séparés en trois catégories (*Figure* 2.5) [CAT 09-HAL 12-RAD 13-RAT 17-KUM 18], la plupart ne s'intéressent qu'à la classification des modules d'un système comme étant sujet ou non à des défauts d'où le nom de classification binaire. La deuxième catégorie s'intéresse à trouver le nombre d'erreurs que pourrait contenir une classe, méthode ou un composant en utilisant la régression [GRA 00-PAI 07-RAT 15]. Enfin, dans de très rare cas, les chercheurs utilisent la classification multiple afin de prédire la sévérité des défauts que pourrait cacher un certain module logiciel et son impact sur le système (High, Medium et low), et ainsi prioriser les instances que le modèle de défaut à jugées très risquées (High) lors de la phase des tests [ZHO 06-SHA 08-SHA 11].

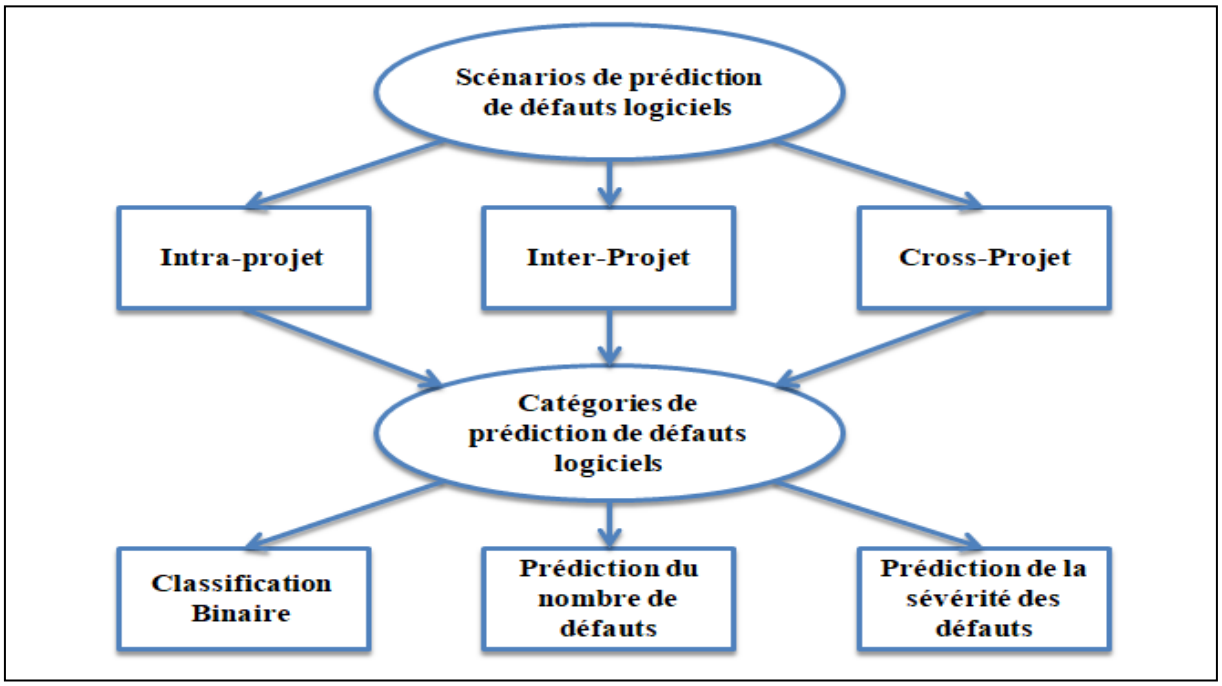


Figure 2. 5- Catégories des travaux SFP

Ces mêmes catégories peuvent s'appliquer sur diffèrent scénarios ou schémas de prédiction de défauts logiciels, qui sont inter, intra et cross-projets expliqués en détails dans la prochaine section.

2.3.3 Scénarios de prédictions de défauts logiciels :

Le choix de la catégorie est souvent vite fait par les chercheurs dans le domaine SFP, où généralement, la classification binaire est employée. Par contre, choisir quel scénario suivre est grandement liée aux données disponibles lors de la recherche [RAD 13-RAT 17 -KUM 18-KHA 21]. Les études de prédiction de défauts logiciels peuvent être séparées en deux branches principales [KHA 21] (*Figure* 2.6), la première est Within-project (au sein du même projet), elle-même devisé en deux scénarios différents, l'intra-projets et l'inter-projet, où les données d'apprentissage et de validation sont tirées et exploitées sur la même ou sur différentes versions du logiciel étudié.

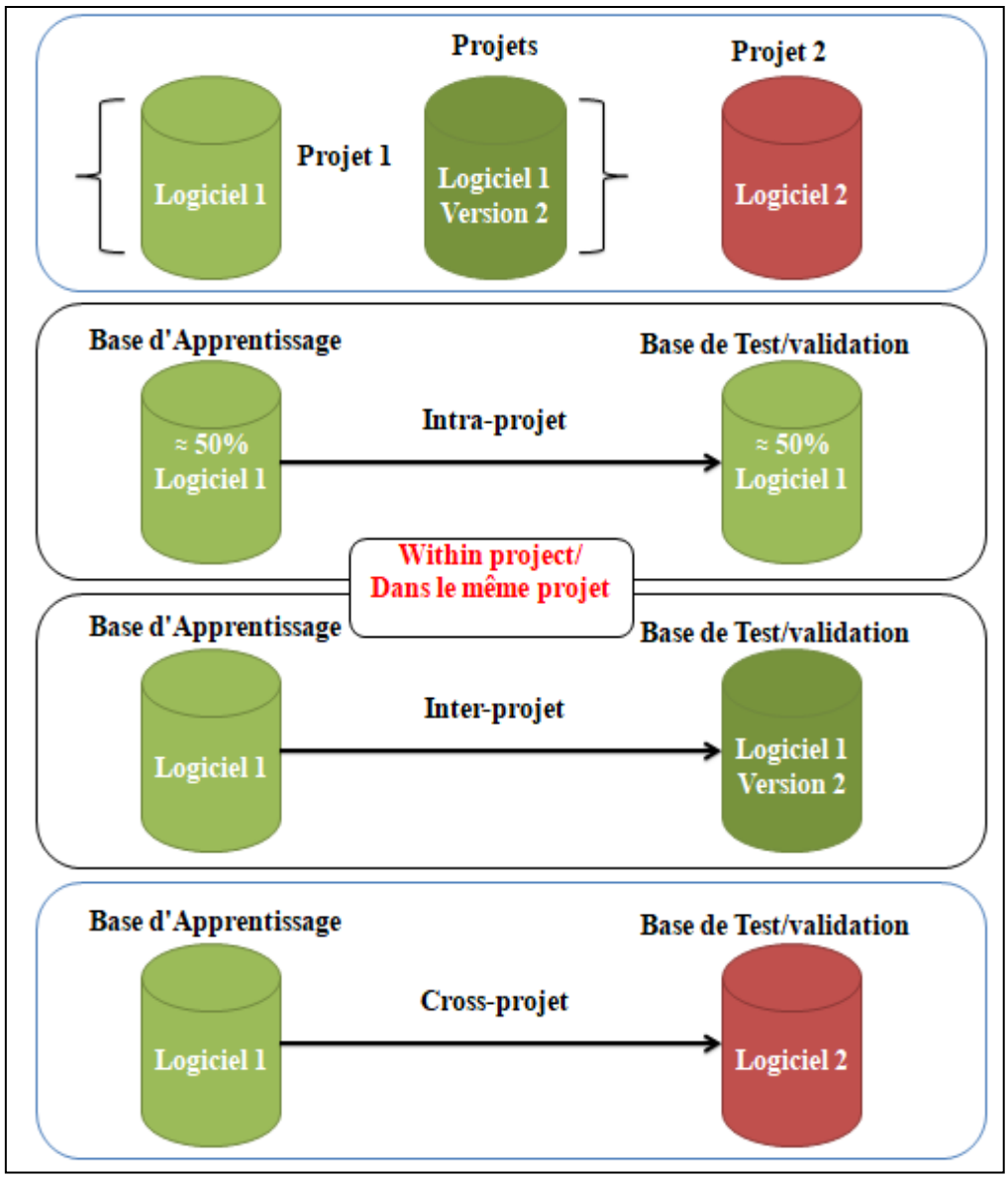


Figure 2. 6- Scénarios/Schémas de prédiction de défauts logiciels

Enfin, le dernier scénario appelé Cross-projets (Cross-project) consiste à construire des modèles de prédiction et les tester à partir de données importées de systèmes complètements différents [ZIM 09-MAL 15-KUM 18]. La suite de cette section va aborder la façon avec laquelle les 3 scénarios de SFP sont conduits dans la littérature.

Prédiction de défauts logiciels Intra-projets (Intra-Project Software Faults Prediction) :

C'est le schéma le plus pratiqué dans le domaine du SFP, portant souvent le terme ***CV*** dans la littérature pour signifier la validation croisée (Cross-validation). Dans ce type d'études, les données d'apprentissages et de tests sont extraites du même logiciel, où une partie du système est utilisée afin de prédire les défauts de la fraction restante [CAT 11-HAL 12-RAT 17-PAN 21]. Généralement, CV utilise la validation croisé (N-fold cross-validation) afin d'évaluer le modèle construit, cette technique divise aléatoirement les données d'apprentissage en N segments (habituellement N=10), N-1 portions sont employées afin de construire le modèle SFP et le fragment restant fait office de données de tests. La validation continue jusqu'à ce que toutes les parties jouent le rôle d'instances d'apprentissage et de tests. Dans des cas très rares [CAT 09-HAL 12], les données relatives aux défauts sont simplement séparées en deux parties, 60% pour l'apprentissage et le reste pour la validation (tests).

Étant le scénario le plus populaire, de nombreuses méthodes de classifications ont été utilisées telles que : Random Forest (RF) [CAT 07- KAU 15], Naive Bayes (NB)[MEN 07], Arbres de décisions (J48) [JIA 08], Raisonnement à partir de cas (CBR) [KHO 03], Support Vector Machines (SVM) [ELI 08-ALT 17], Réseaux de neurones [ERT 15-MAN 18] et Régression Logistique (LR) [KAU 15-KHO 03]. Cette dernière est la méthode la plus utilisée dans ce domaine.

Prédiction de défauts logiciels Inter-projets (Inter-Project Software Faults Prediction) :

Après une lecture approfondie de la littérature SFP, l'Inter-projets (or Cross-release/Previous version) est le champ d'étude de prédiction de défauts logiciels le moins exploité, bien que ce scénario corresponde parfaitement à la définition de la prédiction de défaillance système citée précédemment, car la majorité de la recherche en SFP est centré sur CV dans la branche within-project [MAL 16-RAT 17]. Communément abrégé PV pour Previous version [HER 18], où les données d'apprentissage et de tests sont importées de deux versions différentes du même projet (logiciel). Dans l'exemple de la *Figure* 2.6, les données relatives aux erreurs dans ***le logiciel 1*** sont utilisées pour construire le modèle de prédiction des défauts, par la suite testé par les instances de la version 2 du même système.

Fréquemment, les chercheurs du domaine combinent plusieurs versions du même programme afin de construire le modèle PV, puis ils le valident avec les données d'une autre version du logiciel indépendamment de l'ordre des sorties [HER 18]. Pendant ce temps-là, d'autres travaux comme celui de Malhotra [MAL 16] utilisent une approche par pair (Pairwise) afin d'évaluer les modèles de prédiction de défauts inter-système. Dans ce cas-ci, les données d'apprentissages sont tirées d'une version du logiciel et testés directement par les instances de la version qui le suit.

Prédiction de défauts logiciels Cross-projets (Cross-project Software Faults Prediction) :

Dans les deux schémas précédents, nous supposons la disponibilité de données permettant la construction des modèles SFP. Cependant, le plus souvent, les instances relatives aux

défauts (historique logiciel) ne sont pas vraiment collectées ou disponibles, surtout si c'est la première version du système, sans oublier l'ajout d'une nouvelle tâche au cycle de vie d'un logiciel pour maintenir à jour ces données [ZIM 09-RAT 17-KUM 18-KHA 21]. Ainsi, dans la dernière décennie, un nouveau scénario a été introduit, où, cette fois, les données sont importés de deux projets différents, que ce soit dans la même société ou non, appelé la prédiction de défauts Cross-projets, qui est la meilleure traduction du terme (Cross-Project Defects Prediction "CPDP"). Par conséquent, si les données d'apprentissages sont extraites du logiciel 1 elles seront forcément validées par les instances du programme nommé logiciel 2 comme le montre l'exemple de la *Figure* 2.6.

Herbold et al [HER 16] ont identifiés 3 façons pour appliquer le paradigme CPDP dans la littérature :

- **Par paire (Pair-wise CPDP) :** à chaque entrée dans les données d'apprentissage, une seule base de données est sélectionnée du même projet, puis tester par toutes les bases ou versions des autres programmes une par une.
- **Stricte (Strict-Project Defect Prediction**): toutes les bases de données en dehors du projet cible (données de test) ou une version du programme visé sont utilisées afin de construire le modèle de prédiction des défauts.
- **Mixte (Mixed-Project Defect Prediction)** : ressemblant à la méthode précédente. Mais cette fois, il est permis d'inclure des données originaires des versions antérieures au programme cible lors de la phase d'apprentissage. Un cas plus extrême est le fruit des travaux de Turhan et al [TUR 11] où certaines des instances du logiciel visé sont aussi sélectionnées comme données d'apprentissage.

Même si le scénario CPDP semble attrayant et connait un grand essor dans la communauté de prédiction de défauts logiciels ces dix dernières années, ce n'est pas évident de montrer qu'il est possible d'introduire ce schéma lors de développement d'un système. Zimmermann et al [ZIM 09] ont montré qu'un simple changement de langage de programmation nuit aux performances du modèle construit. De plus, ils ont démontré l'absence de symétrie, par exemple les données extraites du logiciel FireFox ont facilement prédit les défauts que pourrait contenir le programme Internet-Explorer, cependant le contraire n'est pas vrai.

Nous sommes en train de parler des données d'apprentissages et tests, sans mentionner de quoi sont fait ces historiques logiciels et quels sont leurs rapports avec les défauts ? La réponse à ces questions se trouve dans la section suivante.

2.4 Les Métriques Logicielles (Software Metrics)

L'historique logiciel utilisé dans les bases de données propre à la prédiction de défauts logiciels, n'est composé, comme le titre la section l'indique, que de métriques logicielles qui sont le résultat du processus appelé mesure logicielle (Software Measurement). Dans cette section nous allons définir ce qu'est une métrique ainsi que le but de la mesure logicielle. Enfin, nous allons exposer quelle sont les métriques les plus utilisées dans le domaine SFP.

2.4.1 Les Mesures Logicielles (Software Measurement)

Les Mesures représentent des éléments essentiels dans tous les domaines scientifiques ainsi que les disciplines d'ingénierie et font partie intégrante de la vie quotidienne. Elles décrivent les entités étudiées en fonction des normes prédéfinies [FEN 97], en assignant des chiffres et symboles aux attributs qui les composent. Ainsi, elles permettent l'acquisition d'informations qui peuvent être utiles pour développer de nouvelles théories, des modèles de conception et, surtout, évaluer la qualité du produit fourni [MOR 01-KAN 03].

Software Measurement (SM) est un processus quantifiant objectivement et empiriquement les caractéristiques du système logiciel, donnant des informations relatives sur l'état du programme tout le long du cycle de vie logiciel [SCH 92-FEN 00-MOR 01-KAN 03-GEN 05-NAI 08]. Par conséquent, ces caractéristiques vont permettre au manager de mieux contrôler le développement du système, établir le coût, ainsi qu'à mieux allouer les ressources disponibles, ce qui va résulter par un produit final respectant les temps et le budget initial [LIN 08-TIA 05-ABR 10].

Enfin, elles permettent aussi à découvrir des défauts sous-jacents et intervenir le plus tôt possible pour régler ces problèmes en impactant ainsi, considérablement la qualité du logiciel. Ces mesures, à juste titre, sont considérées comme des indicateurs de Software Quality [FEN 00-MOR 01-GEN 05-NAI 08].

Une mesure (métrique) est le mapping d'une entité logicielle telle que son code source en un attribut comme sa complexité [FEN 00- MOR 01]. Une métrique peut être simple (le nombre de lignes d'un programme) ou complexe (l'expérience des développeurs impliqués)

Généralement, nous pouvons identifier trois entités relatives au logiciel et leurs attributs qui sont sujets aux mesures [FEN 97-FEN 00-MOR 01-KAN 03] :

- Processus : toute activité reliée au développement du logiciel telle que l'analyse du changement, spécification, conception codage et test.
- Ressource : personnel, matériel et logiciel utilisés lors du développement.
- Produit : C'est un résultat intermédiaire ou final provenant d'un processus logiciel tel que la documentation du système, donnée de test, le code source et le code objet.

Ces entités sont séparées en deux types d'attributs dans la mesure de logiciel [FEN 97-FEN 00-MOR 01-KAN 03] :

- Interne : Mesuré en termes de processus produit, ou ressource, comme : la complexité du système, sa fonctionnalité et sa taille.
- Externe : elles sont mesurées en respectant la relation entre processus, produit ou ressource et son environnement.

Exemple : Maintenabilités du code source d'un programme, coût, compréhensibilité.

Choisir quelles métriques utilisées est tout aussi important, si ce n'est pas plus que le choix des méthodes apprentissages pour construire les modèles SFP, [CAT 09-KIT 10-HAL 12-RAD 13]. Certains peuvent se demander quel est le lien entre ces simples métriques et un

défaut logiciel… Intuitivement, nous pouvons conclure que si un programme comporte un nombre de lignes important, le risque de trouver des erreurs est plus grand, et aussi qu'il sera plus difficile de comprendre ou maintenir ce genre de programmes, impactant la qualité de ce dernier. Un autre exemple, si une partie du système, telle qu'une classe, est développée par un programmeur moins expérimenté qu'un autre dans l'équipe de développement, le risque qu'elle soit sujette aux défauts sera plus élevé. [GRA 00-OST 05-KOR 08-MOS 08-KOR 09-KIT-10].

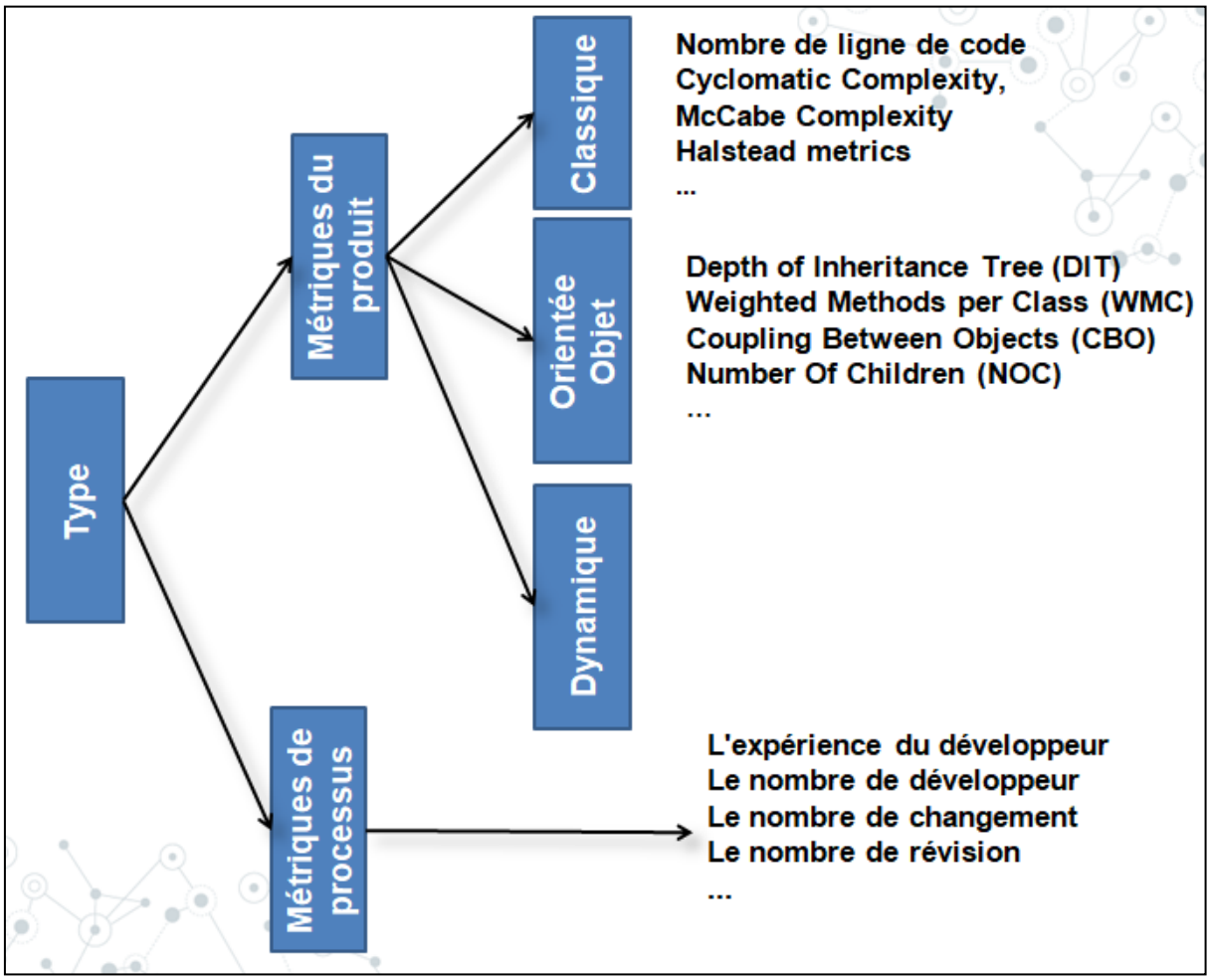


Figure 2. 7- Type de métriques logicielles utilisées en prédiction de défauts logiciel

Nous pouvons distinguer deux grands familles de mesures logicielles (*Figure* 2.7) souvent employées dans la prédiction des modules système sujets aux défauts [CAT 09-RAD 13-RAT 17-KUM 08]:

- **Métriques du produit (Product Metrics) :** Ces mesures sont extraites directement du code source ou lors de l'exécution du système. Ces métriques peuvent être séparées en trois types qui sont classique, orientée objet et enfin dynamique.
- **Métrique de processus (Process Metrics) :** Elles sont le produit du processus de développement lui-même, tel que le nombre de changements qu'a subis le programme, le nombre de développeurs impliqués ...

Dans le reste de cette section, nous allons exposer la majorité des métriques généralement utilisées dans les travaux du domaine de la prédiction de défauts logiciels.

2.4.2 Métriques du produit (Product Metrics)

Parmi les métriques les plus utilisés dans cette catégorie, nous trouvons la complexité de McCabe et les métriques de Halstead. Ainsi que les métriques orientés objet (OO) de Chidamber et Kemerer (CK) et surtout le nombre de lignes de code (LOC) qui représente la taille d'un module (Size). Ces métriques dépendent du code source (static code metrics) afin d'être calculées. Une autre particularité de ces mesures est qu'elles sont efficaces lorsque le système n'est pas déployé [RAD 13].

Complexité cyclomatique de McCabe (CC) :

La complexité cyclomatique (Cyclomatic complexity) a été développée par Thomas McCabe en 1976 [MCC 76]. Cette mesure provient de la théorie des graphes, où le nombre cyclomatique indique le nombre de régions dans un graphe. Telle qu'appliquée dans le GL, c'est le nombre de chemins linéairement indépendants que comprend un programme. Elle est surtout utilisée pour estimer l'effort requis pour tester le programme. Pour déterminer les chemins, le code du logiciel est représenté comme un graphe fortement connexe avec une entrée et une sortie unique. La formule pour calculer CC est la suivante :

$$\boldsymbol{V(G) = e - n + 2p} \qquad \textbf{(2.1)}$$

- V (G) est le nombre cyclomatique du graphe G du programme,
- e est le nombre d'arcs,
- n est le nombre de nœuds,
- p est le nombre de parties du graphe non connexe. Si le graphe est fortement connexe, ce nombre est nul.

Fenton et Ohlsson [FEN 00 A] ont rapporté que la métrique CC est un indicateur raisonnable quand il s'agit de prédire quel module logiciel est sujet aux défauts. Les travaux de Munson et Elbaum [MUN 98] ainsi que ceux de Xu et al [XU 07] ont encouragé l'utilisation de CC dans ce genre de problème vu les résultats obtenus dans leurs études. Par contre, Menzies et al [MEN 02] clame que la complexité cyclomatique ne donne pas de résultats significatifs pour cibler les erreurs.

Métriques de Halstead (MHAL) :

C'est une suite de métriques proposées par l'américain Halstead [HAL 77] dans les années 70, c'étaient les premières mesures exclusives aux logiciels. Ici le programme est composé d'un ensemble d'opérateurs et d'opérandes représentant les expressions simples du système (exemple : a+b/ a et b sont des opérandes, + c'est l'opérateur). Ces métriques sont basées sur quatre nombres primitifs :

- *n1 le nombre total d'opérateurs uniques dans un programme,*

- *n2 le nombre total d'opérandes uniques dans un programme,*
- *N1 le nombre total d'opérateurs dans un programme,*
- *N2 le nombre total d'opérandes dans le programme*

Ces quatre nombres vont servir à calculer 4 autres métriques de bases afin d'évaluer les propriétés d'un programme :

- n = n1 + n2 le vocabulaire d'un programme
- N = N1 + N2 la taille d'un programme
- V = N log n1 le volume d'un programme.
- N = n1 log n1 + n2 log n2 la taille estimée d'un programme (log de base 2).

A partir de ces 4 mesures de bases, Halstead [HAL 77] a déduit plusieurs autres métriques logicielles :

- **Le Niveau de difficulté (D) :** c'est l'aptitude du programme à subir des défauts, si les mêmes opérandes sont utilisés plusieurs fois dans le programme, le système aura de forte chance de contenir des erreurs.

$$\boldsymbol{D = \frac{n1}{2} * \frac{N2}{n2}} \quad \textbf{(2.2)}$$

- **le Niveau du programme(L) :** cette métrique stipule que plus le système est de bas niveau plus il sera défaillant.

$$\textbf{L= } \boldsymbol{\frac{1}{D}} \textbf{ (2.3)}$$

où D représente le résultat de l'équation 2.2

- **L'effort de développement (E)** : Halstead [HAL 77] propose aussi une métrique qui estime l'effort fournis afin d'implémenter le logiciel.

$$E = V * D \quad (2.4)$$

Où D représente le résultat de l'équation 2.2, et V est le volume du programme.

Dans la littérature de la prédiction de défaillances logicielles, la suite de métriques Halstead donne des résultats peu probants, Khoshgoftaar et Allen [KHO 01], ainsi que Menzies et al [MEN 02] s'accordent à dire que ces métriques ne sont pas adaptées à ce genre de problèmes. De plus Menzies et Di Stefano [MEN 04] déclarent que la métrique CC donne de meilleurs résultats que ce de MHAL.

Le nombre de lignes de code (LOC) :

Aussi loin que s'étendent les études SFP, la métrique qui représente la taille (size) du programme est toujours présente. La taille du système est calculée par le nombre de lignes de code (LOC "Line of code "). Bien que la définition de cette métrique soit simple, plusieurs chercheurs tentent de trouver le meilleur moyen de représenter la taille du logiciel [RAD 13-RAT 17-KUM 18], par exemple l'inclusion ou non des commentaires.

De manière générale, les travaux qui utilisent LOC omettent les commentaires ou toute ligne vide dans le code du système. Les caractéristiques de la mesure LOC sont :

- NCLOC : le nombre de lignes de code sans commentaire.
- CLOC : le nombre de lignes des commentaires.
- LOC : c'est La taille physique totale où LOC = NCLOC + CLOC, nous pouvons ainsi mesurer la densité de commentaires par CLOC/LOC. LOC peut être plus pratique à utiliser que NCLOC, de plus, elle est plus facile à extraire.
- CHAR : Le nombre de caractères, c'est une autre mesure de la taille physique qui peut être une base pour le calcul de LOC, LOC = CHAR/a où a est le nombre moyen de caractères par ligne de texte.

Dans l'ensemble, les travaux vérifiant la capacité de LOC à prédire les défauts classent les modules du logiciel selon leur taille, puis ils étudient si les grands ou les petits composants sont les plus susceptibles de contenir des erreurs [RAD13]. Zhang [ZHA09] a examiné si la taille des entités du programme au niveau package sur 3 projets Eclipse à la moindre relation avec les défauts. Les résultats ont montré que 20% des plus grands modules sont responsables de plus 51% des erreurs trouvées. Dans le même principe, Ostrand et al. [OST 05] ont étudies LOC sur deux grands projets industriels, ils ont trouvé en moyenne que 73% des bugs se manifestent dans 20% des modules qui comptent le plus de lignes de code.

Koru et al [KOR 09] déclarent que les modules avec une taille réduite sont plus problématiques que les plus larges, même si le nombre de défauts augmente de manière très faible avec l'incrémentation de LOC. Généralement, la métrique LOC à une très grande affinité avec la présence de défauts, la rendant très utile pour la prédiction des défauts logiciels.

Les métriques Orientées Objet (OO) :

L'essor de la programmation orientée objets a rendu les métriques précédentes dépassées, excepté parfois LOC. En effet, les métrique CC et Halstead sont plutôt adaptées à des systèmes développés en utilisant le paradigme procédural [CAT 09 a- BEE 10-RAD 13-MAL 15-RAT 17-KUM 18]. Ainsi, le besoin de nouvelles mesures s'est fait sentir. Ces métrique sont appelées métriques Orientée objets (OO).

Les métriques le plus utilisées et populaires dans ce cas sont la suite de mesures de Chidamber et Kemerer [CHI 91-CHI 94], souvent abrégée C&K (CK metrics). Leur popularité est non seulement due au nombre de travaux qui les utilisent, mais aussi la disponibilité de logiciels qui facilitent grandement leur extraction (tels que le plugin metrics de Eclipse [http://metrics.sourceforge.net/]). Les métriques CK les plus connues sont [CHI 94]:

1. **Depth of Inheritance Tree (DIT):** Profondeur de la classe dans l'arbre d'héritage En cas d'héritage multiple, DIT est égale à la profondeur maximale depuis la classe racine jusqu'à la classe mesurée.

2. **Weighted Methods per Class (WMC) :** Somme pondérée des complexités des méthodes d'une classe, il y a 3 façons de la calculer [RAD 13]:
    - WMC simple : c'est le nombre de méthodes d'une classe, chaque méthode à une complexité égale à 1.
    - WMC_LOC : comme son nom l'indique, la complexité de la méthode est calculée par rapport au nombre des lignes de code qui la constituent, ainsi $WMC\_Loc = \Sigma^{n}_{n=1}(\text{MLOC})$ où MLOC est le LOC d'une méthode.
    - WMC_CC: c'est la somme des complexités cyclomatiques (CC) de toutes les méthodes de la classe.
3. **Coupling Between Objects (CBO): C'**est le nombre de classes auxquelles une classe est couplée. Deux classes sont dites couplées si une méthode de l'une utilise une méthode ou un attribut de l'autre.
4. **Number Of Children (NOC) :** Nombre de descendants immédiats de la classe dans la hiérarchie de classes.
5. **Response For a Class (RFC) :** c'est la cardinalité de l'ensemble de réponse d'une classe. C'est l'ensemble des méthodes qui peuvent être directement appelées lors de l'exécution de n'importe quelle méthode de cette classe.
6. **Lack of Cohesion in Methods (LCOM) :** c'est la différence entre le nombre de paires de méthodes qui n'accèdent pas aux mêmes attributs et le nombre de paires de méthodes qui accèdent aux mêmes attributs d'une classe. Elle est égale à 0 si le résultat est négatif.

Plusieurs travaux se sont acharnés à étudier la relation entre les métriques OO et les défauts [BAS 96-BRI 00-EL 01-GYI 05-OLA 07-SHA 08-SIN 10-KAU 15-KUM 18-DOS 21]. Leurs résultats montrent que même si CK métriques sont très effectives pour indiquer les défaillances, toutes ne sont pas aussi performantes. Ainsi, les meilleures mesures CK selon les travaux cités précédemment, sont CBO, WMC et RFC. Par contre, DIT et NOC ne sont pas de bons indicateurs [TAN 99-SIN 10]. Enfin, la métrique LCOM donne des résultats plutôt mitigés et ne s'avère efficace que si le projet ou ses modules sont de taille modeste [BAS 96-OLA 07-SHA 08].

Dans la littérature présente et avec la disponibilité de base de données open source, d'autres mesures orientés objets deviennent de plus en plus populaires, car, dans ces bases, elles accompagnent toujours les métriques CK. Une liste non exhaustive de ces métriques OO est fournie dans la *Table* 2.1.

***Table* 2. 1-Métriques Orientée objets dans les travaux SFP**

NOM	Signification	Source
LCOM3 **Lack of Cohesion in Methods**	Afin de le calculer il faut utiliser l'équation suivante : $\mathbf{LCOM3} = \frac{(\frac{1}{a}\sum_{j=1}^{a}\mu(Aj)) - m}{1-m}$ avec	Henderson [HEN 96]

	m : nombre de méthodes par classe. a : nombre d'attributs par classe μ(A) : nombre de méthodes qui accèdent à l'attribut A.	
Ca **Afferent couplings**	Représente le nombre de classes qui dépendent de la classe mesurée.	Martin [MAR 94]
Ce **Efferent couplings**	Représente le nombre de classes dont la classe mesurée est dépendante.	Martin [MAR 94]
NPM **Number of Public Methods**	Le nombre de méthodes d'une classe qui sont déclarés comme publiques. La métrique est connue aussi comme Class Interface Size (CIS)	Métriques QOOD [BAN 02]
DAM **Data Access Metric**	Le ratio entre le nombre d'attributs privés (ou protégés) sur le nombre total d'attributs de classe.	
MFA **Measure of Functional Abstraction**	Cette mesure est le ratio du nombre de méthodes héritées par une classe sur le nombre total de méthodes accessibles par la méthode de la classe. Les constructeurs sont ignorés.	
MOA **Measure of Aggregation**	Elle mesure l'étendue de la relation partitive (partie-tout), réalisée en utilisant les attributs. Cette mesure est le nombre de champs de classe dont les types sont des classes définies par le développeur.	
CAM **Cohesion Among Methods of Class**	Cette mesure calcule la connexité entre les méthodes d'une classe sur la base de leurs paramètres. La métrique est calculée en utilisant la somme des différents types de paramètres de chaque méthode divisée par le produit du nombre de différents types de paramètres des méthodes de toute la classe et le nombre de méthodes.	
IC **Inheritance Coupling**	Cette mesure fournit le nombre de classes parents auxquelles une classe donnée est couplée. Une classe est couplée à sa classe parent, si l'une de ses méthodes héritées est fonctionnellement dépendante de la méthode redéfinie ou nouvelle de la classe. Une classe est couplée à sa classe parent si l'une des conditions suivantes est remplie : • L'une de ses méthodes héritées utilise un attribut qui est défini dans une nouvelle méthode. • L'une de ses méthodes héritées appelle une méthode redéfinie. • L'une de ses méthodes héritées est appelée par une	Tang [TAN 99]

	méthode redéfinie et utilise un paramètre qui est défini dans une méthode redéfinie.	
CBM **Coupling Between Methods**	Le nombre total de méthodes nouvelles ou redéfinies auxquelles toutes les méthodes héritées sont couplées. Il y a un couplage si au moins l'une des conditions définies dans métrique IC est satisfaite.	
Average Method Complexity **AMC**	Cette métrique mesure la taille moyenne des méthodes pour chaque classe. La taille d'une méthode par exemple : est égale aux nombres de codes binaires Java dans la méthode.	

Olague et al [OLA 07] ont rapporté que les métrique QOOD sont appropriées à l'étude de la construction de modèles de prédiction de défauts logiciels, même si leurs performances sont en-dessous des mesures *CK*. Briand et al [BRI 99] ont déclaré que la métrique *Ce* est plus efficace que la métrique *Ca, et* qui sont toutes les deux le fruit de la séparation de la métrique CBO en deux parties. El Emam et al [EL 01 A] ont montré que la combinaison entre LOC et les métriques OO est très corrélée avec les défauts. Enfin, les métriques orientées objets sont souvent très efficaces d'après la littérature SFP, Radjenović et al [RAD 13] stipulent qu'elles sont d'autant plus performantes que la prédiction de défauts est faite avant le déploiement du logiciel.

2.4.3 Métriques de processus (Process Metrics)

Représentant la deuxième grande famille de métriques logicielles utilisées dans la prédiction de défauts logiciels même si elles sont moins communes que le groupe précédent. Les mesures de type processus (MDP) sont généralement calculées à partir de l'historique des changements que peut subir un logiciel ou l'historique entre au moins deux versions du système [HAL 00-RAD 13-SHE 14].

Ces métriques sont séparées en deux catégorie Delta et code churn (attrition)[HAL 00]. Estimées pour toutes les métriques, Delta est le fruit entre la variation des valeurs des métriques entre deux itérations du programme. Un exemple simple c'est l'ajout de nouvelles lignes de code. Cependant, si le même nombre de lignes ont été ajoutées et retirées, la valeur Delta restera le même et l'information sera perdue, ainsi le changement ne se verra plus. D'où le besoin d'une autre façon de s'y prendre qui est le code churn qui capture tous les changements dans les métriques qui ont eu lieu entre les deux itérations [MUN 98-RAD 13]. Les mesures de processus qui reviennent le plus dans la littérature sont :

- **Le nombre de développeurs :** désigne le nombre développeurs qui sont responsables de la modification du fichier.
- **L'expérience de développeur** : se rapporte à l'expérience acquise par développeur mais elle est rarement utilisée.
- **Âge du module** : c'est l'âge concret du module inspecté (code).

- **Le nombre de changements apportés** : c'est le nombre de changements apportés à la nouvelle version.
- **Le changement de la taille d'un programme** : exprimé en LOC.
- **Le nombre de défauts passés** : nombre de défauts déjà corrigés.

Munson et Elbaum [MUN 98] ont rapportés que code churn MDP ont une très grande corrélation avec les bugs détectés en comparaison avec Delta. En plus, ils ont déclaré qu'il n'y a pas de relation entre le nombre de développeurs implémentant les changements et les troubles décelés.

Moser et al [MOS 08] ont comparés 18 MDP à des métriques classiques, leurs résultats indiquent que les métriques de processus recèlent des informations significatives qui aident à repérer les défauts mieuxx que les mesures tirées du code source lui-même. Dans un autre papier, Schröter et al [SCH 06] ont cherchés à déterminer si un développeur spécifique pourrait ou non insérer plus de défaillances qu'un autre. D'après eux, même s'il y des différences sur la densité des défauts entre programmeurs, elle ne serait pas liée à leurs compétences mais plutôt à la complexité du code à écrire.

En outre, les études [KHO 00-LAY 08-AMD 09-ILL 10-MAT 10-RAD 13] déclarent toutes que les modèles de prédiction des défauts logiciels construits en employant les métriques MDP performent mieux que les mesures tirées du code source quand le système est déjà déployé. Enfin, Moser et al [MOS 08] expliquent pourquoi les MDP performent aussi bien par le fait que le code source est écrit par des développeurs qualifiés, Alors que ce sont les changements que le système subit au cours du temps qui introduisent les erreurs que les métriques de processus capturent mieux que les mesures OO par exemple.

Même si nous n'avons pas abordé toutes les métriques utilisées dans les 30 ans de littérature SFP, nous nous sommes concentrés sur les plus récurrentes d'entre elles. Par contre une question se pose ; Comment estimer les performances des modèles de prédiction de défauts logiciels construits en utilisant ces métriques ? La réponse se trouve dans la section suivante.

2.5 Évaluation des modèles de prédiction de défauts logiciels :

Avant de s'étaler sur la façon avec laquelle les modèles de prédiction de défauts sont évalués, nous allons introduire une notion supplémentaire [JIA 08-MAL 15-KUM 18] qui est les échantillons.

Un échantillon (instance) : c'est un ensemble fini d'exemples afin que l'algorithme arrive à construire son modèle d'apprentissage (apprendre). Dans le cas de la prédiction de défauts logiciels, une instance représente une classe accompagné par les valeurs de ses métriques calculées, ainsi que l'information relative aux défauts (Généralement *True* si la classe comporte des défauts, false si elle n'est pas sujette aux erreurs) (*Figure* 2.8).

En théorie il existe trois types d'échantillons :

1. **D'apprentissage** : pour la création du modèle.

2. **De validation** : pour optimiser le modèle.
3. **De test** : pour vérifier la qualité des modèles.

Nom	WMC	CBO	RFC	LCOM	NPM	LCOM3	LOC	AMC	défauts
QuartzExchange	4	26	33	6	4	2	176	5.5	True
MinaComponent	6	6	8	15	2	2	26	28.333333	False

Figure 2. 8- Un exemple d'échantillon dans la prédiction de défauts logiciels

Par contre, en pratique seules les instances d'apprentissages et de tests sont prises en charge. Bien entendu, comme nous sommes dans l'environnement de la prédiction de défauts logiciels, la méthode de validation choisie dépend grandement du scénario SFP étudié, où les échantillons d'apprentissages et tests sont séparés selon les schémas décrits dans la section 2.3.3.

Evaluer les modèles de prédiction reste à ce jour un débat continuel dans communauté SFP. Une large exploration de la littérature a montré qu'il n'existe aucun consensus sur la meilleure mesure afin d'estimer les performances des algorithmes d'apprentissages utilisés pour construire le modèle de prédiction de défauts [JIA 08-BEE 10-CAT 12-MAL 15-HER 18-RAT 17-KUM 18]. Un exemple représentatif de cette discorde est rapporté par Herbold et al [HER 16], où 5 papiers de prédiction de défauts ont été présentés dans la 37ème conférence internationale en génie logiciel (37th International Conference on Software Engineering) tenue en mai 2015[PRO 05]. Aucun d'eux n'a employés exactement les mêmes métriques de performance, rendant le choix de la mesure à utiliser dépendante des préférences des auteurs [RAT 17].

Les métriques de performance peuvent être séparés en deux catégories [CAT 12-RAT 17-KUM 18]:

1. ***Mesures numériques (Numeric measures):*** les plus utilisés dans la littérature, elles donnent une description numérique des performances des modèles de prédiction étudiés. Nous pouvons citer Puissance (accuracy), Précision (Precision), Rappel (Recall), F1 (f-measure), taux de faux positifs (false positive rate), G-means, taux de faux négatifs (false negative rate), J-coefficient, Spécificité (specificity) ...
2. ***Mesures graphiques*** : utilisées pour afficher des informations de manière simple et compréhensible. Largement dominé par la courbe ROC et, à un degré moindre la courbe Précision/Rappel dans la littérature.

Presque toutes les mesures trouvent leurs origines dans la matrice de confusion ci-dessous :

Table 2. 2- Matrice de confusion (Confusion Matrix)

	Sujet au défaut	**Non Sujet au défaut**
Sujet au défaut	**TP**	**FN**
Non Sujet au défaut	**FP**	**TN**

Où :

- ***TP :*** le nombre de vrais positifs qui représente le nombre d'exemples sujets aux défauts qui sont classés comme tels.
- ***FN :*** le nombre de faux négatifs qui représente le nombre d'exemples étiquetés fault prone mais classés faux par le modèle.
- ***FP :*** le nombre de faux positifs (faux rejet), exemples non sujets aux défauts mais classés vrais.
- ***TN :*** le nombre de vrais négatifs, qui représente le nombre d'exemples sans défauts classés comme tels par le modèle.

A partir des nombres ci-dessus, nous pouvons ainsi calculer nos métriques de performances :

- **Erreur de type 1** = $\frac{FN}{TP+FN}$ appelé aussi taux de fausse alarme
- **Erreur type 2** = $\frac{FP}{FP+TN}$ appelé taux de faux rejet.
- **Taux d'erreur** = $\frac{FN+FP}{\mathrm{TP+FN+TN+FP}}$
- **Le taux des exemples bien classés (Puissance / Accuracy)** = $\frac{TP+TN}{TP+FP+TN+FN}$
- **Rappel (Recall « PD »)** = $\frac{TP}{TP+FN}$ Mesure la probabilité qu'un module sujet aux pannes soit correctement classé.
- **Précision (Precision « PR »)** = $\frac{TP}{TP+FP}$ montre le nombre de modules prédits correctement comme sujets aux défauts parmi tous les modules prédits défectueux.
- **Spécificité** =$\frac{TN}{TN+FP}$
- **Anti-Spécifité= 1-Spécificité** = $\mathbf{1} - \frac{TN}{TN+FP} = \frac{FP}{FP+TN}$ (taux de faux positifs)
- **F-mesure (mesure F/F1-score)**= $\mathbf{2} * \frac{(\boldsymbol{PD}*\boldsymbol{PR})}{(\boldsymbol{PD}+\boldsymbol{PR})}$

Selon plusieurs travaux, la puissance (accuracy) n'est pas un bon indicateur quand il s'agit de comparer des classifieurs pour la prédiction, ceci étant lié à un problème inhérent dans le domaine SFP. Ce problème est que les bases de données avec lesquelles les modèles de prédiction sont construit sont des données déséquilibrées (imbalanced Data), où le nombre des instances non sujets aux défauts est plus grand que celles qui contiennent des erreurs [JIA 08-LES 08-BEE10-HAL 12-CAT 09-MAL 16-RAT 17-KUM 18-PAN 21].

Une solution à ce problème est d'utiliser d'autres métriques comme la Précision ou le Rappel, qui abritent en leurs seins des informations très pratiques d'un point de vue professionnel [CAT 12-KAU 15]. En effet, un grand score en PD vaudra dire que les modèles arrivent à prédire le plus possible de modules défectueux. D'un autre côté, plus le résultat

affiché par la métrique Précision est élevé, plus l'effort de tests sera réduit [CAT 12-RAT 17-KUM 18].

D'autre travaux tendent à utiliser d'autres méthodes d'évaluation, plus précisément tirées des mesures graphiques telle que l'Aire sous la courbe ROC (AUC "Area Under The Receiver Operating Characteristic curve")[CAT12-MAL 16-ERT 15]. Dans le cas d'un classifieur binaire, il est possible de visualiser les performances sur cette courbe. La courbe ROC est une représentation du taux de vrais positifs (Recall) en fonction du taux de faux positifs (anti-spécificité) dépendant d'un certain seuil qui varie de 1 à 0. La courbe ROC commence au point (0, 0) jusqu'à (1, 1). Son intérêt est de s'affranchir de la taille des données de test quand les données sont déséquilibrées, ce qui est toujours le cas dans les travaux SFP. Cette représentation met en avant un nouvel indicateur qui est l'aire sous la courbe **(AUC).**

Une courbe est dite idéale si elle passe par le point de (0, 1) et avec l'aire sous la courbe AUC = 1, indiquant ainsi qu'il n'y a pas d'erreur de prédiction Une valeur AUC acceptable pour une courbe ROC doit être supérieure à 0.5. Dans le cas où AUC ne dépasse pas les 0.5, on dit que le modèle n'est pas meilleur qu'un classifieur aléatoire [JIA 08-ZHO 10-ERT 15].

Jusqu'à maintenant, nous savons quels sont les acteurs principaux pour la construction d'un modèle de prédiction de défauts logiciels et comment les évaluer. Par contre, nous ne savons pas quelle est l'origine des bases données d'apprentissage et de validation des méthodes ML. La section suivante va s'attarder sur les bases de données utilisées dans la littérature SFP, ouvrant ainsi la porte à tous ceux qui veulent se lancer dans le domaine.

2.6 Base de données pour la prédiction de défauts logiciels (Defects Datasets) :

Comment nous l'avons abordé, les instances des bases de données utilisées dans la prédiction des pannes de logiciels sont composées de métriques logicielles ainsi que des informations relatives aux défauts comme le montre la *Figure* 2.8. Ces données sont généralement tirées de l'historique logiciel. Cet historique est souvent géré en exploitant un système de gestion de versions tel que CVS (Concurrent Versions System). Le moindre changement ou correction, par exemple, dans une classe (programmation orientée objet) entre deux versions du même programme, conduira le chercheur à étiqueter l'échantillon comment étant sujet aux défauts (*Figure* 2.9) [CAT 09-BEE 10-CAT 11-HAL 12-MAL 15-RAT 17-KUM 18-PAN 21].

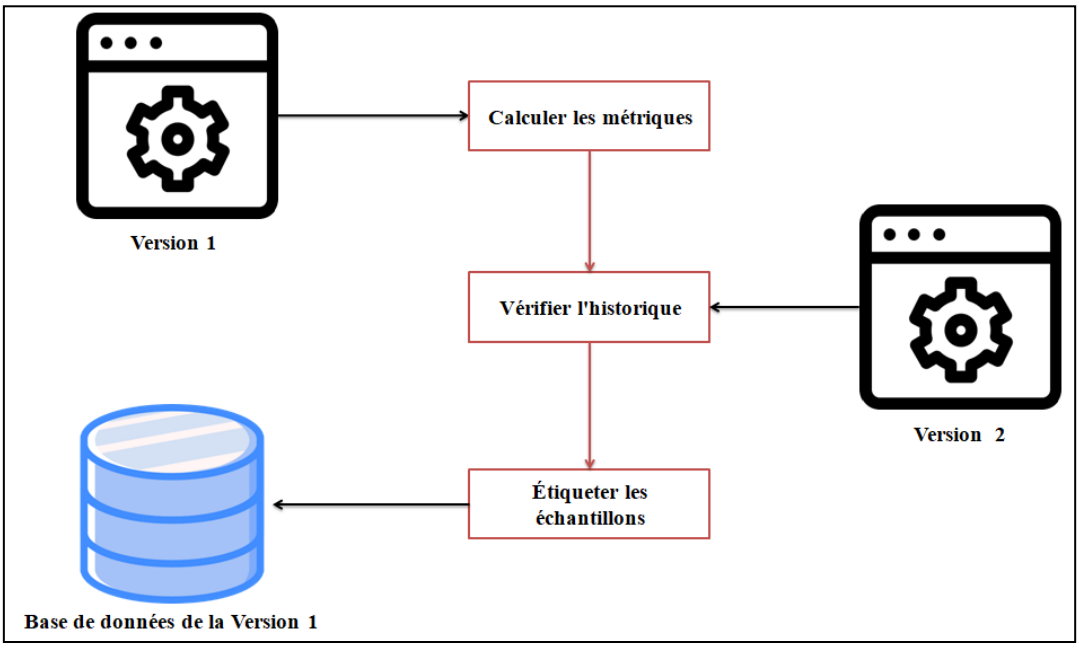
Une chose est sûre, même à ce jour, la disponibilité des données pour la construction des modèles de prédiction est relativement faible, car maintenir ce genre de données est ardu, en ajoutant une nouvelle tâche à un cycle de vie déjà lourd [RAD 13-MAL 15-RAT 17 -KUM 18]. Par contre, la majorité des travaux actuels sont caractérisés par l'utilisation des instances publiques, en contraste avec des papiers antérieurs dominés par les données privées. Nous pouvons distinguer 4 différents types de sources de données [RAD 13, MAL 15]:

- **Données privées** : sont les plus communément employés dans les premières 20 années du domaine, elles sont issues des grands logiciels de l'industrie et des

télécommunications. Selon plusieurs études, les modèles construits en utilisant ce genre de bases de données ne sont pas très fiables.

- **Données semi-publiques** : consiste généralement en des projets open source, où le code est disponible mais pas les métriques, tel que le projet Eclipse.
- **Données des étudiants** : dans ce cas, ce sont les étudiants des chercheurs responsables de l'étude qui ont développés les programmes utilisés comme données [KAN 07]
- **Données publiques** : Les métriques sont complètement disponibles afin de construire des modèles généraux, comparables et fiables. Les bases les plus importante sont PROMISE [MEN 16] et celles publiées par la NASA [SYA 05].

Figure 2. 9 - Schématisation de la construction des bases de données pour les études SFP



Le passage à utilisation des données publique a commencé après la publication des données par la NASA (National Aeronautics and Space Administration) en 2005 [SYA 05]. Ce passage est encouragé car l'utilisation des bases de données libres rend les travaux facilement vérifiables donc plus fiables [CAT 09-CAT 11]. Ainsi, presque tous les recherches actuelles évitent l'utilisation des bases de données privées, même si nous constatons que les modèles construits n'utilisent plus les larges logiciels industriels qui sont la cible première dans la recherche SFP [HER 16-HER 18-RAT 17].

Maintenant que nous avons une vue générale du domaine de la prédiction de défauts logiciels, une présentation de l'état de l'art s'impose. La suite de ce chapitre, va se concentrer à la présentation des quelques travaux intéressants dans le domaine, plus encore, nous exposerons quelle sont les méthodes d'apprentissages bio-inspirées, ou non, les plus populaires dans la prédiction des pannes systèmes.

2.7 État de l'art de la prédiction de défauts logiciels

Après les sections précédentes, nous avons une vue générale du domaine de la prédiction de défauts logiciels. Ainsi, nous savons quelles sont les méthodes utilisées, l'origine des données et de quoi sont composés les bases de données utilisées pour la construction des modèles de prédictions. Tout le monde s'accorde à dire que les algorithmes les plus performants dans ce domaine sont les techniques d'apprentissage automatique qu'elles soient supervisées ou non supervisées, par rapports aux systèmes de classification statistiques [CAT 09- BEE 10- CAT 11-HAL 12- RAD 13- MAL 15- KAU 15- RAT 17- KUM 18-SHA 19-ALI 20-PAN 21].

Table 2. 3- Sélection d'algorithmes utilisés dans la prédiction de défauts logiciels

Modèles	Algorithmes	Sources
Plus proches voisins	• K-plus proche voisin KPPV(IBK)	[RYU 15-HAM 19]
Réseaux de neurones	• Le perceptron multicouche (MLP) • Les réseaux RBF (fonction à base radiale) • Deep learning (CNN/MLP)	[KAN 07-ERT 15-KUM 16-ALQ 20]
Machine à vecteurs de support (SVM)	• SVM • Lagrangian • Least Squares SVM • Linear Programming • Voted Perceptron	[SIN 09-AL 11-RAJ 18]
Arbres de décision	• C4.5 (J48) • Arbre de régression (DTR)	[JIA 08-VAN 08-RAT 17 A]
Théorème de Bayes	• Naive Bayes(NB)	[RAT 16-MAL 16]
Méthodes combinées	• Forêts d'arbres décisionnels (Random Forest) (RF) • Modèle d'arbre logistique • Boosting • Bagging	[KAU 08-AKO 17-RHM 20]
Statistiques	• Régression logistique (LR) • Negative binomial • Univariate Régression logistique (ULR)	[WEY 10-SHA20-RAT 21]

Ainsi, de nombreux travaux s'orientent vers l'utilisation des méthodes ML pour la validation de leurs études. Bien entendu, certains algorithmes sont plus populaires que d'autres, tel que les réseaux de neurones (MLP) ou la Régression Logistique (LR). LR est la méthode de classification statistique la plus utilisée parmi la riche panoplie d'algorithmes d'apprentissage ou statistiques employés dans la littérature SFP. La *Table* 2.3 présente les algorithmes les plus populaires utilisés dans le domaine SFP avec quelques références

intéressantes à ce sujet. Enfin, nous pouvons voir une synthèse des travaux dans la *Table* 2.5. Dans cette synthèse, nous avons utilisé des abréviations afin de la faciliter sa lecture (OO: Métrique orientée objet/MDP: Les mesures de type processus/MCC : Complexité cyclomatique ...) Le sens de chaque abréviation a été déjà présenté dans ce chapitre.

2.7.1 Travaux en Prédiction de Défauts Logiciels Intrar-projets (CV)

Les méthodes d'apprentissage automatique utilisées ne sont pas juste réservées aux algorithmes supervisés, certains travaux ont relevé le challenge d'appliquer des méthodes ML non supervisées (clustering)[ZHO 04-CAT 09 A-BIS 11] . Zhong et al. [ZHO 04], ont utilisé les méthodes K-moyennes (K-means) et Neural-Gas clustering pour regrouper les modules, puis un expert ayant 15 ans d'expérience en ingénierie a marqué chaque cluster par fault prone ou non, tout en donnant le niveau de sévérité de chacun des groupes étiquetés, par exemple : le cluster 1 est fault prone avec 90% de chance, le cluster 2 avec 50% et ainsi de suite.

Catal et al. [CAT 09a] ont appliqué l'algorithme de Clustering des X-moyennes (X-means) avec 6 métriques de type méthode, et, afin de séparer les clusters obtenus, ils ont comparé les métriques de chacun des centroïdes des groupes obtenus avec leurs seuils. Ainsi, ils ont pu étiqueter les clusters en fault prone ou non sans l'intervention d'un expert. Pour valider leur technique, ils ont comparé X-means avec K-means et fuzzy clustering en utilisant des données collectées à partir d'un logiciel privé turque (white-goods manufacturer developing embedded controller software). Les résultats obtenus par X-means montrent qu'ils ont pu construire une technique complètement automatisée, sans même avoir besoin des données des anciens logiciels et sans l'aide d'un expert.

Bien entendu, la majorité des papiers SFP sont dominés par les techniques ML supervisées. Erturk et Sezer [ERT 15] présentent, pour la première fois, l'application des réseaux de neurones adaptatifs à inférence floue (ANFIS) au problème de prédiction de défauts logiciels. De plus, deux autres méthodes qui sont MLP et SVM ont été utilisées et comparées avec ANFIS. Les données utilisées dans cette étude sont recueillies à partir de NASA Dans cette étude, les métriques de complexité de McCabe sont les seules à être employées. ROC-AUC a été utilisé pour mesurer les performances des méthodes. Les résultats obtenus sont comme suit 0,7795, 0,8685, et 0,8573 respectivement pour SVM, MLP et ANFIS. Par conséquent ANFIS semble être une technique prometteuse pour la prédiction des défauts logiciels.

Kaur et Kaur [KAU 15] ont appliqué 17 classifieurs sur un ensemble de 44 projets open source obtenus à partir du référentiel PROMISE. L'aire sous la courbe ROC (AUC) pour chacun des 17 classificateurs est obtenue après les avoir appliqués sur les 44 ensembles de données un par un afin de vérifier la robustesse et la stabilité de ces méthodes. Leurs expériences montrent que les forêts d'arbres décisionnels, la régression logistique et Kstar sont les plus robustes et les plus stables pour la prédiction des défauts de logiciel. En outre, ils démontrent que le classifieur Bayes Naïf et les réseaux bayésiens se sont révélés être des classifieurs ayant une mauvaise stabilité.

Les deux travaux précédents se sont limités à classifier les modules logiciels sujets aux défauts ou non. Par contre, Rathore et Kumar [RAT 16] ont essayé de prédire le nombre de

défauts que peut avoir un module. Ils ont examiné la capacité de arbres de décision par régression (DTR) pour prédire le nombre de défauts dans deux scénarios différents, avant le déploiement et après, pour un système logiciel donné. L'étude expérimentale est réalisée sur cinq projets open-source avec leurs dix-neuf versions recueillies à partir du référentiel de données PROMISE [2]. La précision prédictive de DTR est évaluée en utilisant l'erreur absolue et l'erreur relative. Les résultats montrent que DTR donnent une très bonne précision pour la prédiction du nombre de défauts dans les deux scénarios envisagés.

Lessman et al [LES 08] ont conduits une étude comparative sur 22 algorithmes ML sur 10 bases de données fournis par la NASA [SYA 05], ils ont trouvé que la puissance et robustesse de ces méthodes n'ont pas de différence significative pour choisir quel est le meilleur algorithme à appliquer dans les études SFP. Enfin, Kanmani et al [KAN 07], présentent deux modèles de prédiction de pannes logicielles à base de réseau de neurones multicouches MLP et de réseaux de neurones probabilistes (PNN) avec des métriques orientées objet. Ils les ont empiriquement validés en utilisant un ensemble de données collectées à partir des logiciels développés par leurs étudiants. Les résultats sont comparés avec deux modèles statistiques utilisant cinq attributs de qualité. Les auteurs ont constaté que les réseaux de neurones PNN surpassent MLP pour la prédiction des défauts logiciels.

2.7.2 Travaux en Prédiction de Défauts Logiciels Inter-projets (PV)

Il est très difficile de trouver des papiers intéressent s'attardant exclusivement sur la recherche SFP inter-projets. Comme nous l'avons mentionné précédemment, le scénario PV est le moins populaire parmi les études sur la prédiction des erreurs systèmes [RAT 17].

ZHO et al [ZHO 10] ont examiné la capacité des métriques de complexités telles que MCC à prédire les défauts logiciels au niveau orientée objet. Pour cela, ils ont collecté les données auprès de trois projets Eclipse, de plus ils ont utilisé la méthode de classification "Univariate Logistic Regression". L'objectif principal de ce papier est de répliquer le travail de Olague et al [OLA 08], ces derniers ont conduit leur étude sur six versions du système Rhino. Ils ont trouvé que les métriques de complexité ont de très bonnes performances pour faire la discrimination entre les classes défectueuses et les sans défauts. Par ailleurs, ils déclarent que des métriques moins connus telles que SDMC (Standard deviation method complexity) [MIC 05] et AMC (Average method complexity) donnent de meilleurs résultats que des mesures plus récurrentes comme LOC et WMC. Par contre, Zhou et al [ZHO 10] réfutent cette conclusion, argumentant que la combinassent entre les deux métriques LOC et WMC pour la construction des modèles de prédiction intra ou inter-projets ont de meilleures performances que n'importe quelles métriques de complexité combinée avec LOC, et aussi que les métriques de complexité donnent des résultats très mitigés d'après les données extraites des projets Eclipse.

Harman et al [HAR 14] ont étudié les modèles de prédiction de défauts inter-projets sur 8 bases de données importées de 8 versions consécutives du logiciel Hadoop. Les modèles sont construits par l'algorithme SVM où ses paramètres utilisateurs sont ajustés par l'algorithme génétique (GA). Ainsi, les auteurs affirment que les modèles entraînés par des données

extraites des versions les plus anciennes que celle de la cible (à tester) sont préférables à ceux construits par les versions du système les plus récentes. Entre outre, construire un modèle de prédiction de défauts PV par une version du programme Hadoop puis tester directement par celle qui la suit performe le mieux.

Plus récemment, Malhotra [MAL 16] a comparé 18 méthodes d'apprentissage automatiques sur 7 applications Android et leurs multiples versions pour la prédiction de défauts inter-projets. Les résultats obtenus suggèrent que les réseaux de neurones ainsi que Naïve Bayes (NB) sont très adaptés à ce genre de problèmes. Par contre, les techniques SVM et voted perceptron sont à éviter quand il s'agit de PV.

Dans leurs efforts de prédire le nombre de défauts que pourraient contenir un module logiciel dans un scénario PV par la méthode Decision Tree Regression (DTR), Rathore et Kumar [RAT 16] ont procédé en sélectionnant 19 bases de données représentant 5 projets open source du référentiel de données PROMISE [MEN 16]. Cette fois, au lieu de suivre le schéma classique de validation par paire, où chaque modèle construit par une version d'un projet est testé par la version qui le suit, les auteurs ont choisi d'utiliser toutes les instances des versions qui précédent celle ciblée pour faire l'apprentissage de la méthode DTR. Les résultats ont démontré que l'algorithme DTR arrive très bien à prédire le nombre d'erreurs que pourraient contenir un module du système, et que les performances de la technique sont meilleures sur toutes les bases sélectionnées dans un environnement intra-projets par rapport à ceux produits par les modèles inter-projets.

Enfin, Juneja [JUN 19] a proposé d'intégrer la logique floue (fuzzy logic) pour construire des modèles SFP inter-projets. Ainsi, les méthodes Gain et Gain Ratio sont utilisées pour savoir quelles sont les métriques qui impactent le plus la découverte d'erreurs. Puis, des règles floues sont appliquées pour extraire les variables indépendantes pour la construction du modèle de prédiction de défauts logiciels à partir de 11 projets open sources trouvés dans PROMISE. À la fin de ce processus, ils ont appliqué le classifieur Neuro-fuzzy pour filtrer les données d'apprentissages et de tests. Les auteurs s'accordent à dire que ces étapes ont produit des résultats plus efficaces que ceux des algorithmes référencés dans le domaine tel que Random Forest (RF), MLP, SVM et Naïve Bayes selon plusieurs mesures de performances.

2.7.3 Travaux en Prédiction de Défauts Logiciels Cross-projets (CPDP)

Comme nous l'avons mentionné auparavant, la rareté des données disponibles pour la construction des modèles SFP, plus la difficulté d'entretenir ce genre de données ont conduit à l'établissement de ce nouveau scénario de prédiction de défauts logiciels. Le Cross-projets (CPDP) s'intéresse à l'utilisation des instances d'apprentissages tirées d'un projet autre que celui ciblé par le testeur [ZIM 09-RAT 17]. Depuis dix ans, le CPDP connaît un très grand essor dans la communauté SFP, il faut dire que l'idée de construire des modèles de prédiction pour un nouveau projet sans se préoccuper de la disponibilité des données ou de l'entretenir est très attrayante quoique difficile. Car malgré toutes ces années de recherche, il n'y pas de preuve tangible que le cross-projets fonctionne [RAT 17-PAN 21].

Un des travaux pivots dans ce domaine a été mené par Zimmermann et al [ZIM 09]. Dans leur démarche, ils ont proposé un critère de performance où la Précision, Recall et la puissance doivent être égales ou supérieures à 0.75 pour que le modèle construit soit déclaré performent. Ainsi, ils ont exécuté 622 paires de modèles de prédictions sur 28 bases de données tirées de 12 projets open sources et commerciaux. Les résultats de leurs expériences montrent que seulement 21 paires (3.4%) ont satisfaits le critère établi. D'autant plus, il s'avère que les modèles CPDP ne sont pas vraiment symétriques, car là où les données extraites du logiciel Firefox ont réussi à prédire les défauts du système Internet Explorer, le contraire est très loin de cela.

Turhan et al [TUR 09] ont proposé d'utiliser le NN-filtre pour ne sélectionner que les échantillons pertinents dans une base de données, en se basant sur la similarité entre les données filtrées les instances de tests. Bien que les auteurs rapportent qu'ils ont réussi à réduire le taux de fausse alarme, les performances restent loin de celles de CV.

D'un autre côté, He et al [HE 15] ont fait une investigation sur la construction des modèles de prédiction de défauts à partir d'un nombre très réduit de métriques (variables indépendantes). L'expérience fut conduite sur 34 bases de données tirées de 10 projets open sources se trouvant dans PROMISE. Les auteurs affirment qu'un modèle CPDP construit avec seulement 5 variables indépendantes (donc métriques logicielles) donnent des performances acceptables, ainsi qu'une méthode d'apprentissage telle que le Naïve Bayes (NB) performe mieux dans ce cas de figure. L'année d'après, Herbold et al [HER 16] ont étudié les effets d'extraire les données locales sur les performances des modèles CPDP stricts. Ils ont utilisé l'algorithme de clustering WHERE sur les données d'apprentissages et tests. Après, ils ont appliqué la méthode SVM sur chaque cluster pour évaluer les performances des modèles locaux. Enfin, les résultats montrent qu'il n'y pas d'évidence positive pour l'utilisation d'un tel procédé.

Plus récemment, Herbold et al [HER 18] ont conduit une étude colossale en comparant 24 approches Cross-projets proposés dans la littérature. Dans cet effort, ils ont répliqué ces travaux en n'utilisant que des données open sources. À leur surprise, la meilleure approche fut celle de Cruz et Ochimizu [CRU 09], qui est une des premières approches proposées pour CPDP. Cruz et Ochimizu [CRU 09], ont transformé les données d'apprentissage et de test en utilisant la fonction logarithme et la médiane comme référence. Leur but était de montrer le problème de variations des métriques entre différents projets.

Notre état de l'art ne s'arrête pas juste à cette section, la suite va se focaliser sur les méthodes de Data Mining bio-inspirées, même si nous en avons cités quelque unes précédemment.

2.8 Méthodes bio-inspirées dans la prédiction de défauts logiciels :

La diversité biologique a conduit à une multitude de méthodes bio-inspirées de fouille de données. Bien entendu, la littérature SFP regorge de travaux utilisant ces algorithmes, certaines techniques sont plus populaires que d'autres telles que les réseaux de neurones et

l'algorithme génétique (GA) surtout en comparaison aux méthodes intelligence en essaim (Swarm Intelligence) ou les systèmes immunitaires artificiels (AIS). Par ailleurs, les méthodes bio-inspirées ne se limitent pas qu'à la classification des modules d'un logiciel en tant que sujets ou non à des défauts. Ces méthodes peuvent être utilisées pour trouver les meilleures valeurs des paramètres utilisateurs d'une autre technique d'apprentissage afin d'améliorer ces performances. Un exemple de ce procédé est présenté par Harman et al [HAR 14], où ils ont utilisé l'algorithme génétique pour ajuster les paramètres de la méthode SVM. Une autre possibilité pour l'utilisation des méthodes d'inspiration biologique est sélection de caractéristique (feature selection), où le but cette fois est de trouver un sous-ensemble de variables indépendantes pertinente. Dans le cas SFP, cela consiste en la sélection des métriques logicielles qui ont une forte relation avec les défauts pour une certaine base de données. Cette dernière fonctionnalité est plus ou moins le centre d'intérêt des méthodes biologique, si on oublie les réseaux de neurones. La *Table* 2.4 présente quelques algorithmes bio-inspirés de fouille de données dans le domaine de la prédiction de défauts logiciels triées par popularité.

***Table* 2. 4-Sélection d'algorithmes bio-inspirés utilisés dans la prédiction de défauts logiciels**

Modèles	Algorithmes	Sources
Réseaux de neurones	• le perceptron multicouche (MLP) • les réseaux RBF (fonction à base radiale) (NET RBF) • Deep learning	[KAN 07-ERT 15-KUM 16-ALQ 20]
Métaheuristique	• Algorithme Génétique (GA)	[SAR 12-HAR 14]
Swarm Intelligence	• Optimisation par essaims particulaires (PSO) • Algorithme de colonies de fourmis (ACO)	[ABD 15-KUM 18 A]
Système Immunitaire Artificiels (AIS)	• Immunos-81 • Sélection clonal (CLONALG/CSCA) • AIRS	[CAT 09 B-ABE 14]

Pour vérifier les facteurs influents sur les performances des méthodes d'apprentissage automatique dans la prédiction de défauts logiciels, AL Qasam et al [ALQ 20] ont utilisé deux méthodes neuronales de Deep learning qui sont le perceptron multicouches, ainsi que le réseau neuronal convolutif (Convolutional Neural Networks CNN). Les modèles construits dans cette étude sont basés sur des métriques de niveau procédural telles que LOC et MCC, extraites de 4 bases de données open source fournies par la NASA [SYA 05]. Le résultat de leur recherche montre que trouver les paramètres utilisateur optimaux pour une méthode de classification donnée, joue un rôle prépondérant dans l'amélioration de ses performances.

Comme nous l'avons mentionné quelques lignes auparavant, la sélection de caractéristiques (feature selection) est l'objective principale de la majorité des travaux utilisant des méthodes biologiques. À cet égard, Wahono et Herman [WAH 14] ont procédé en sélectionnant les métriques pertinentes (variables indépendantes) de 9 bases de données NASA composés de 22 à 37 mesures de niveau procédural (MCC, Halstead, LOC) par deux méthodes bio-inspirées. Ces algorithmes sont l'algorithme génétique (GA) ainsi que PSO (Particle Swarm Optimization). Pour évaluer les performances de GA et PSO, ils ont construit des modèles de prédictions par 10 techniques ML. Après, ils ont comparé ses modèles avec ceux construits par les données traitées par les deux algorithmes biologiques (après la sélection de caractéristiques). D'après une analyse des résultats par le test statistique t-test, leur expérience montre une amélioration des performances des 10 méthodes d'apprentissages choisies. Par contre, il n'y pas de différence significative entre les performances de GA et PSO pour la sélection de caractéristiques.

Un cas particulièrement intéressant fruit du travail de Abdi et al [ABD 15], c'est l'utilisation de la méthode PSO pour la classification et non seulement la sélection de variables. Pour cela, ils ont étudié la prédiction de défauts logiciels basés sur les règles d'apprentissages par une version modifiée de l'algorithme Optimisation par essaims particulaires (PSO) car la méthode PSO n'est pas faite pour fonctionner dans un environnement où les données sont codées de façon discrète, ce qui est le cas dans toutes les bases de données liées au domaine SFP. Dans ce papier, les auteurs ont appliqué la méthode de clustering K-means sur 4 bases de données de la NASA, puis ils ont extrait les règles d'apprentissages par la méthode DSMOPSO (Distance-based Multi-objective Particle Swarm Optimization). Le résultat établi par leur expérience montre que leur modèle de prédiction performe mieux que des algorithmes d'apprentissages référencés tels que MLP en présence de large base de données.

Le chapitre 1 montre notre intérêt pour les méthodes de Data Mining inspirées des systèmes immunitaires et, dans cette section, nous nous concentrons plus particulièrement sur leur utilisation dans la prédiction de défauts logiciels. Par rapport à d'autres algorithmes bio-inspirés, les travaux utilisant les systèmes immunitaires artificiels (AIS) sont très rares, on ne dénombre que 5 papiers malgré 30 années d'études dans ce domaine. L'un de ces travaux a été mené par Catal and Diri [CAT 09 B], les auteurs ont étudiés l'effet de la taille de l'ensemble de données, métriques et des techniques de sélection de caractéristiques sur les problèmes de prédiction des pannes logicielles. Cette fois, ils ont utilisé 6 algorithmes AIS (AIRS 1,2 et parallèles, CLONALG, Immunos 1 et 2) ainsi que 3 algorithmes d'apprentissage automatique bien connus sur cinq ensembles de données publics de la NASA. Les résultats de leur étude ont montré que RF offre les meilleures performances de prédiction pour les grandes bases de données et NB est meilleur pour les plus petites. AIRS2 Parallel est le meilleur algorithme basé sur le paradigme des systèmes immunitaires artificiels lorsque les métriques de niveau procédural sont utilisées. En ce qui concerne les métriques OO, Immunos-2 est le meilleur.

Avant de terminer ce chapitre, nous allons exposer certains outils utilisés dans les études de prédiction de défauts logiciels, afin d'encourager de nouveaux chercheurs à s'investir dans

ce domaine de recherche. Les logiciels les plus importants sont présentés dans la section suivante.

2.9 Outils utilisés dans la prédiction de défauts logiciels

Comme tout domaine de recherche, la recherche SFP dépend grandement de sa communauté, même si les premiers travaux restent discrets à propos des outils utilisés dans leurs papiers. L'essor de l'apprentissage automatique a joué un grand rôle dans la popularité de la prédiction de défauts logiciels. Ainsi, un transfert naturel de logiciels employés dans le Machine Learning à la SFP s'est fait facilement. Bien entendu, certains langages de programmation sont plus utilisés que d'autres à cause, encore une fois, de l'activité de leurs communautés comme le langage **JAVA** et **Python**. Les outils qui sont souvent cités dans la littérature SFP sont :

- **WEKA (Waikato environment for knowledge analysis)** [5-HOL 94-HAL 09-WIT 16] : un logiciel open source distribué sous la licence GNU (General Public License), il fournit un ensemble de classes et d'algorithmes en Java implémentant les principaux algorithmes de data mining. C'est l'outil le plus utilisé car il implémente presque tous les algorithmes de classification statistique et d'apprentissage automatique.
- **MATLAB** [6] : est un langage de programmation de quatrième génération émulé par un environnement de développement du même nom. Il est utilisé à des fins de calcul numérique. Développé par la société The MathWorks, MATLAB permet de manipuler des matrices, d'afficher des courbes et des données, de mettre en œuvre des algorithmes, de créer des interfaces utilisateurs, et peut s'interfacer avec d'autres langages comme le C, C++, Java, et Fortran. Les utilisateurs de MATLAB sont de milieux très différents comme l'ingénierie, les sciences et l'économie dans un contexte aussi bien industriel que pour la recherche. Matlab peut s'utiliser seul ou bien avec des boîtes à outils.
- **KEEL (Knowledge Extraction based on Evolutionary Learning)** [7-ALC 09-ALC 11-TRI 17]: Même s'il n'est pas aussi connu que les deux autres. Nous voulions l'intégrer à cette liste, car cet outil écrit en Java et plus encore open source, contient un catalogue fourni de méthodes d'apprentissage automatiques et de sélection des caractéristiques bio-inspirées, tels que ACO et PSO ainsi que plusieurs Algorithmes évolutionnistes (AE).
- **La bibliothèque Tensorflow** [8-ABA 16]: la bibliothèque ultime quand il s'agit des méthodes de réseaux de neurones, plus encore les algorithmes de Deep Learning écrit en Python. Sachant l'engouement actuel sur les techniques d'apprentissages profond, Tensorflow et le langage Python par association deviennent actuellement incontournables dans le domaine ML, et dans un futur proche, la communauté de prédiction de défauts logiciels.

L'utilisation des outils open source est devenue incontournable. Même si au début, le logiciel MATLAB était le plus employé, le regard actuel de la littérature penche vers le

programme WEKA, et presque tous les travaux de ce domaine s'en servent, afin de produire des papiers plus vérifiables et répétables.

Avant de terminer, notons que nous avons plus ou moins utilisés tous les outils cités au-dessus. Nous nous sommes concentrés finalement sur WEKA car il contient la bibliothèque implémentant les systèmes immunitaires artificielles (AIS) que nous avons vu dans le chapitre 1.

Table 2. 5 - Synthèse des travaux en prédiction de défauts logiciels

Source	Scénario	Méthodes	Données	métrique	Description	Résultats
LI 93	CV	LR	Privée	CK	Le premier papier étudiant la capacité des métriques CK pour la prédiction des défauts. Sur deux logiciels commerciaux.	Presque toutes les métriques OO étudiés sont des bons indicateurs de défauts sauf LCOM
BRI 01	CV	LR	Privée	OO	Le but c'est d'explorer la relation entre les métriques OO et les défauts.	Très peu d'information sont fournis sur les performances des modèles construit, mais les auteurs affirment qu'il y a une grande corrélation entre les métriques OO et les défauts.
ZHO 04	CV	K-means N-Gas	NASA	MCC MHAL LOC	Utilisation des méthodes ML non supervisées et analyse les résultats par un expert en QL	L'expert a réussi à étiqueter les clusters produis par les deux algorithmes. Un exemple : si votre module se trouve dans le cluster 1 il a 90% d'avoir des défauts
KAN 07	CV	MLP PNN	Projets d'étudiants	CK	Présentent deux modèles SFP en utilisant MLP et de réseaux de neurones probabilistes (PNN) sur un ensemble de données collectées à partir des logiciels développés par leurs étudiants.	Dans ce cas de *Figure* PNN donne de meilleurs résultats par rapport au réseau de neurones multicouche dans la prédiction de défauts logiciels
LES 08	CV	22 méthodes ML (MLP, J48,SVM, RF, ...)	NASA	MCC MHAL LOC	Une étude comparative sur 22 algorithmes ML sur 10 bases de données fournis par la NASA	Les auteurs constate qu'il n'y a pas de différence significative pour déclarer qu'elle méthode est plus adapté pour les études SFP.
OLA 08	PV	ULR	Privée	MCC	Examiner la capacité des métriques de	Ils ont trouvés que les métriques de complexité ont

				OO	complexités tell que MCC à prédire les défauts logiciels au niveau orientée objet. Sur six versions du système Rhino.	de très bonnes performances pour faire la discrimination entre les classes défectueuses et les sans défauts et que des métriques moins connus tell que AMC donnent de meilleurs résultats que des mesures plus récurrentes comme LOC et WMC.
CRU 09	CPDP	LR	Publique	CK	Ils ont transformé les données d'apprentissages et de tests en utilisant la fonction logarithme et la médian comme référence sur le système Mylyn.	La transformation de données améliore grandement les performances des modèles CPDP toute en réduisant les variations des métriques entre différant projets.
ZIM 09	CPDP	LR	¨Privée Publique	MDP MCC	Ils ont proposé un seuil de performance où le Rappel, la Précision et la puissance dévoient dépasser les 75% pour dire que le modèle arrive à prédire les défauts. Leur étude est conduite sur 28 bases de données, Extraites de projets open sources tell qu'Eclipse, d'autre privée comme Internet Exploreur (IE).	Sur 622 modèles construits et validés par pairs dans un scénario CPDP, il n'y a que 21 pairs qui remplissent le critère proposé. Un autre fait constaté, c'est que la prédiction de défauts CPDP n'est pas symétrique. Les données de Firefox arrivent très bien à prédire les pannes d'IE, mais le contraire est faux.
CAT 09 A	CV	X-means	Privée	MCC MHAL LOC	Utilisation des méthodes ML non supervisées. Et comparé les centroïde des groupes obtenus avec un seuil fixé par les auteurs	La méthode X-means performe mieux que d'autre algorithme du genre (K-means et fuzzy cluster). Ça conduit à une approche complètement automatisée pour SFP.
CAT 09 B	CV	6 méthodes AIS MLP NB J48 RF	NASA	MCC MHAL LOC CK	les auteurs ont étudiés l'effet de la taille de l'ensemble de données, métriques et des techniques de sélection de caractéristiques sur les problèmes de prédiction des pannes logicielle	RF est la meilleure méthode présence de larges bases de données et NB pour les petites. AIRS2.parallèle performe mieux quand les métriques sont MCC,MHAL et LOC. Par contre, Immunos-2 est meilleur lorsque les

						métriques sont OO.
TUR 09	CPDP	NN-filtre NB	NASA Privée	MCC MHAL LOC	Ils ont utilisés le NN-filtre pour sélectionner que les échantillons pertinents dans une base de données, en se basant sur la similarité entre les données filtrés les instances de tests, le tout sur 7 bases données NASA ainsi que 3 systèmes Privée d'une compagnie Turque de développement logiciel.	Après l'application de NB sur les données filtré, les autres ont trouvé une nette amélioration des performances des modèles CPDP. Par contre, ces performances restent loin des modèles intra-projets, suggèrent de commençais par le cross-projets puis de tout de suite passé à CV si une compagnie ne de possède pas de données pour appliquer SFP dans leur cycle de vie logiciel.
ZHO 10	PV	ULR	Publique	MCC OO	L'objectif principal de ce papier est de répliquer le travail d' Olague et al [OLA 08] sur des bases de données open source tirées de 3 itération du logiciels ECLIPSE	Ils ne sont pas d'accord avec la conclusion d'OLA 08 et ils ajoutent que la combinassent entre les deux métriques LOC et WMC pour la construction des modèles de prédiction CV ou PV ont de meilleures performances que n'importe quelles métriques de complexité combinée avec LOC
LAM 11	CV	NB IBK SVM multinomial NB	Publique	-	Le but c'est de prédire la sévérité des défauts où les modèles sont construits à partir des rapports de bug sur des données extrait des logiciels ECLIPSE, GNOME.	La méthode NB s'avère est la meilleure parmi la sélection des algorithmes étudiés pour le texte mining.
HON 12	CV	RF	Privée	Métrique SDL	Étudier les performances de la méthode d'apprentissage RF pour la prédiction de défauts sur seulement 2 bases de données.	Selon l'auteur RF est plus adapté pour le domaine SFP que d'autres méthodes bien connues comme MLP et SVM.
CAN 13	CPDP	Multi-objective LR GA	PROMISE	OO	Le but de ce papier c'est d'utiliser une approche multi-objective pour la prédiction de défauts	Même si les performances des modèles CPDP sont faible en comparaison à CV. L'approche multi-objective CPDP donne de

					CPDP sur 10 bases de données tirées de PROMISE.	résultats plus compétitifs.
WAH 14	CV	10 méthodes ML GA PSO	NASA	MCC MHAL LOC	Les auteurs ont étudié l'impact de la sélection des caractéristiques par les algorithmes GA et PSO.	Bien que la sélection des caractéristiques améliore les performances des modèles de prédiction. Il n'y a pas de différence entre les performances de PSO et GA.
HAR 14	PV	GA SVM	Publique	LOC CK	Les modèles sont construits par l'algorithme SVM où ses paramètres utilisateurs sont ajustés par l'algorithme génétique (GA). Sur 8 bases de données importées de 8 versions consécutives du logiciel Hadoop.	Construire un modèle de prédiction de défauts PV par une version du programme Hadoop puis testé directement par celle qui la suit performe le mieux. De plus, plus les données utilisées sont loin de celle ciblés, plus l'impact de la prédiction est faible.
ABD 15	CV	PSO K-means	NASA	MCC MHAL LOC	Le but des auteurs est d'utiliser la méthode PSO pour la classification grâce aux règles d'apprentissages et une version modifiée de l'algorithme étudié (Distance-based Multi-objective Particle Swarm Optimization)	D'après les valeurs de la mesure de performance AU, DSMOPSO est capable de donner de meilleures performances sur les 4 bases de données fournis par la NASA que des algorithmes bien connus tell que MLP, RF ...
HE 15	CPDP	J48 NB LR SVM Decision *Table* Bayesian Network	NASA	OO	Le but c'est de trouver le nombre minimal de variables indépendantes (métriques) pour construire un bon modèle de prédiction CPDP, sur 34 bases de données PROMISE	Les auteurs affirment qu'un modèle CPDP construit avec seulement 5 variables indépendantes (donc métrique logiciels) donnent des performances accep*Table*s, ainsi qu'une méthode d'apprentissage tell que le (NB) performe mieux dans ce cas de *Figure*.
ERT 15	CV	ANFIS	NASA	MCC	L'application des réseaux de neurones adaptatifs à inférence floue (ANFIS) au problème de prédiction	Selon la mesure de performance AUC, ANFIS est meilleurs que les algorithmes MLP et SVM

					de défauts logiciels.	
KAU 15	CV	17 méthodes ML (MLP, J48,SVM, RF, ...)	PROMISE	OO	La vérification de la robustesse et la stabilité de 17 algorithmes ML sur 44 bases de données extraites de PROMISE.	RF, LR et Kstar sont les méthodes les plus s*Table*s et robustes, par contre NB et bayésiens se sont révélés être des classifieur ayant une mauvaise stabilité.
RAT 16	CV+PV	DTR	PROMISE	OO	Prédire le nombre de défauts que pourrai contenir un module logiciel par l'algorithme des arbres de décision par régression (DTR) sur 19 bases de données PROMISE	DTR arrive à prédire le nombre de défauts que pourrai contenir un module logiciel avec plus ou moins de précision avant et après le déploiement du programme.
MAL 16	PV	18 méthodes ML	Publique	OO	Comparés 18 méthodes d'apprentissages automatiques sur 7 applications Android et leurs multiples versions pour la prédiction de défauts PV.	MLP ainsi que NB sont très adaptés à ce genre de problèmes. Par contre, les techniques SVM et voted perceptron sont à évités dans quand il s'agit de PV.
HER 16	CPDP	WHERE SVM	NASA PROMISE Publique	OO MCC MHAL MDP	Les auteurs cherchent à étudier les effets d'extraire les données locales sur les performances des modèles CPDP stricte. Les données d'apprentissage et tests sont mélangées puis séparées en deux groupes par la méthode de clustering WHERE	Les résultats montre que il n' y pas d'évidence positive pour l'utilisation d'un tel procédé pour la prédiction de défauts logiciels cross-projets.
YOH 17	CV	Bagging J48	NASA	MCC MHAL MDP	Étudier la capacité des méthodes d'apprentissages combinés ELA pour la prédiction de défauts logiciels sur 8 bases de données de la NASA	L'utilisation des méthodes ELA, après avoir traité les données par un algorithme de sélection des caractéristiques donne des meilleurs résultats que d'utiliser toutes les métriques (variables indépendantes)
HER 18	CPDP	C 4.5	NASA	OO	Les auteurs ont conduit une étude colossale en	Il s'avère que la meilleure approche proposée selon

		NET RBF SVM NB RF NB	PROMISE Publique	MCC MHAL MDP	comparant 24 approches CPDP proposés dans la littérature, mais toutes les données utilisées sont publiques.	leur benchmark est un des premiers travaux dans ce domaine de recherche SFP par Cruz et Ochimizu [CRU 09].
JUN 19	PV	Neuro-fuzzy	PROMISE	OO	L'intégration de la logique floue (fuzzy logic) pour construire des modèles SFP sur 11 projets open sources trouvés dans PROMISE, les règles floues sont appliquées pour extraire les variables indépendantes pertinentes.	Les auteurs s'accordent à dire que ces étape ont produits des résultats plus efficaces que ceux des algorithmes référençais dans le domaine tel que Random Forest (RF), MLP, SVM et NB
ALQ 20	CV	MLP CNN	NASA	MCC MHAL LOC	L'application des méthodes de deep learning pour la SFP et voir qui impacte le plus les performances des modèles de prédiction sur 4 bases de données fournis par la NASA.	Trouver les valeurs optimales des paramètres utilisateurs d'un algorithme d'apprentissage automatiques, joue un rôle important dans les performances des modèles de prédiction.
HAS 21	CV	Whale Optimization Algorithm (WOA)	PROMISE	OO	Les auteurs ont utilisé une nouvelle version de l'algorithme d'apprentissage par essaim de baleines WOA pour la sélection des caractéristiques sur 17 bases de données extraite du référentiel de données PROMISE	L'addition de l'opérateur de sélection naturel à l'algorithme WOA à grandement amélioré les performances de la méthode surtout pour échapper de l'optimal local, ainsi qu'à rendre les modèles construits plus performants.

2.10 Conclusion

Ce chapitre long mais nécessaire, s'intéresse à la prédiction de défauts logiciels (SFP) et ce qu'elle apporte au cycle de vie d'un programme. Pour résumer, la prédiction de défauts logiciels consiste à construire des modèles de prédiction par des méthodes d'apprentissage automatiques à partir de données extraites de l'historique logiciels, plus précisément des métriques logicielles, afin de prédire les modules (class/méthode/composant) d'un nouveau système comme étant sujets aux défauts ou non. Par conséquent, cela permettra au testeur de

cibler les parties du logiciel qui pourraient nuire à son fonctionnement, et ainsi réduire le temps et le budget élevés de cette phase du développement tout en produisant un logiciel de qualité dans les délais impartis.

La suite de cette thèse va exposer nos propositions d’utilisation des méthodes immunitaires artificielles dans la prédiction de défauts logiciels. Plus de détails seront donnés dans le prochain chapitre, mais il faut savoir que, à ce jour, il n'existe que cinq travaux abordant les AIS dans SFP, quatre d'entre eux utilisent le même ensemble de données, mais tous sont conduits dans un scénario intra-projets.

Chapitre 3 : Prédiction de Défauts Logiciels par les Systèmes Immunitaires Artificiels

L'objectif de ce chapitre est de présenter nos premières contributions à la littérature de prédiction des défauts logiciels (SFP). En effet, nous avons commencé notre travail dans ce domaine en menant trois études basées sur les systèmes immunitaires artificiels (AIS)[HAO 17-HAO 17 A-HAO 17 B]. Pour plus de détails sur les AIS, il faut se référer au chapitre 1 section 1.4 et 1.5. Dans le chapitre 2, nous avons exposé toutes les informations jugées nécessaires concernant les modèles SFP. Comme systèmes bio-inspirés, les systèmes immunitaires artificiels (AIS) sont très peu utilisés dans ce domaine, surtout si nous les comparons à des méthodes du genre, tels que les réseaux de neurones ou les algorithmes génétiques. Il est donc intéressant d'investir dans l'étude de leur utilisation dans un champ aussi actif que celui de la prédiction de défauts logiciels. La première partie de ce chapitre porte sur nos contributions dans la prédiction de défauts intra-projets, tandis que la deuxième partie est consacrée à la présentation de l'ébauche d'un outil que nous proposons pour la prédiction de défauts Cross-projets.

3.1 Introduction

Lorsqu'on parle de méthodes bio-inspirées d'apprentissage automatique (ML), tout le monde dans la communauté SFP ou ML pensera aux réseaux de neurones (MLP)car ils ont fait leurs preuves dans plusieurs domaines comme la reconnaissance, la classification et sans nul doute la prédiction de défauts logiciels [KAN 07-AL 11-MAL 16-ALQ 20]. Bien entendu, nous n'allons pas mettre de côté le fait que les êtres humains vont toujours se tourner vers ce qui est familier et populaire, même pour les chercheurs, oubliant par la même occasion les multiples familles d'algorithmes d'inspirations biologiques qui se trouvent dans la littérature. Une de ces familles souvent délaissées est le groupe de méthodes immunologiques (AIS)

Métaphoriquement parlant, un défaut pourrait être vu comme un virus ou une bactérie qui menace l'intégrité du logiciel et son fonctionnement. Par conséquent, identifier (prédire) quelle partie du système représente un danger serait en adéquation avec le rôle des systèmes immunitaires artificiels (AIS). Rappelons que la prédiction de défauts logiciels (SFP), consiste à construire des modèles capables de prévoir quel module du programme est sujet ou non à des défauts par des méthodes d'apprentissages automatiques, par exemple, avant la phase de test. Cela est effectué à partir de données composées essentiellement de métriques logicielles, ainsi que des informations relatives aux défauts tirées de l'historique du programme [CAT 07-LES 08-ZIM 09-BEE 10-CAT 11-HAL 12-RAD 13-HAR 14-MAL 15-KUM 16-RAT 17-KUM 18-SHA 19-ALI 20-RIT 21]. Si le modèle construit s'avérait performant, il occasionnerait une réduction considérable de l'effort de test, que ce soit d'un point de vue

temporel ou budgétaire car le testeur va ne cibler que les parties du logiciel sujettes aux pannes. Ainsi, la qualité du produit délivré sera améliorée.

Après une recherche approfondie dans la littérature des modèles SFP, nous avons pu identifier 5 travaux impliquant l'utilisation des méthodes AIS, résumés dans la Table 3.1. Dans la majorité des cas, le même ensemble de données a été employé pour construire les modèles de prédictions. Ces bases de données sont fournies par la NASA (National Aeronautics and Space Administration) [SAY 05] et sont originaires de plusieurs projets de la fameuse agence. Une autre observation est que tous ces travaux ne s'intéressent qu'à la prédiction de défauts logiciels intra-projets (CV), suggérant qu'il reste tant à explorer et analyser sur l'utilisation des méthodes AIS dans la prédiction des pannes systèmes.

À notre connaissance, le premier travail employant les systèmes AIS dans le domaine SFP est celui qui a été mené par Catal et Diri [CAT 07]. Dans leur initiative, ils ont évalué seulement la méthode AIRS 1 (chapitre 1 section 1.5) sur 5 bases de données NASA contenant des métriques de niveau procédural (chapitre 2 sections 2.4). Selon leurs résultats, quand il s'agit de larges projets, les performances d'AIRS 1 avec l'algorithme de sélection des caractéristiques CFS (Correlation Based Feature Selection) sont les meilleurs. Continuant sur leur lancée, les mêmes auteurs ont cette fois utilisés une seule base de données NASA, composé de 6 métrique orientées objets, plus précisément la suite de mesures C&K (Chidamber and Kemerer [CHI 94]) [CAT 07 A]. Ajoutant la métrique LOC (nombre de lignes de codes) aux 6 C&K mesures, le modèle composé par l'algorithme AIRS 1 à dépasser les performances de la méthode J48, qui est une implémentation en Java du système C 4.5 (arbres de décision).

Deux ans plus tard, Catal et Diri [CAT 09 B] ont conduit leurs investigations sur les effets de la taille des bases de données, le choix des métriques employées et la méthode de réduction de dimensionnalités (Feature selection) appliquées sur les problèmes de la prédiction de défauts systèmes. Ainsi ils ont utilisés 6 AIS méthodes d'apprentissages (AIRS1, 2 et parallèle, CLONALG, Immunos 1 and 2) accompagnées de 3 algorithmes bien connus dans la littérature (Random Forests (RF) , Naive Bayes (NB), J48) sur 5 bases de données fournies par la NASA. La conclusion de leur étude montre que la technique RF performe mieux en présence de larges bases de données et NB quand le nombre d'instances est faible. De plus, AIRS 2.Parallèle est la méthode AIS la plus performante quand les données sont composés de métriques procédurale (MCC, MHAL, LOC voir la section 2.4.2). Par contre, c'est Immunos-2 qui se distingue quand les mesures sont de type OO.

Après les travaux de Catal et Diri [CAT 07-CAT 07 A- CAT 09 B] les méthodes immunologiques sont passées en arrière-plan jusqu'à l'année 2014 avec les travaux d'Abaei et Selamat [ABA 14]. Les auteurs ont refait la même démarche vue dans le travail précédant [CAT 09 B] avec quelques différences, tel que l'ajout des algorithmes CSCA et Immunos-99 à la collection des méthodes AIS. De plus, ils n'ont employé que 4 bases de données NASA où les variables indépendantes sont exclusivement des métriques procédurales (method level). Ces auteurs ont abouti aux mêmes observations que celles déclarées par Catal et Diri [CAT 09 B] concernant la meilleure méthode selon les détails des bases de données et ce, même en

appliquant des méthodes de sélection des caractéristiques. D'un autre côté, les modèles édifiés par la méthode Immunos-99 sont plus performants quand les données sont traitées par un algorithme de réduction de dimensionnalité, contrairement à AIRS2.

Les travaux SFP les plus récents impliquant les AIS ont été menés par Kaur et Kaur [KAU 16]. Bien entendu, le scénario reste le même où les modèles construits sont de nature intra-projet tout comme les papiers résumés ci-dessus. Par contre, il y a une différence appréciable touchant les données et les métriques qui les composent. Ainsi, dans leur démarche les auteurs ont étudié les performances de 10 méthodes d'apprentissages automatiques, 8 techniques AIS et 15 algorithmes évolutionnistes sur des données extraites de 3 sous-projets Eclipse. Les instances de ces bases de données sont composées de métriques de type processus (MDP) appelés "Micro-interaction Metrics (MIM)" collectés par Lee et al [LEE 11]. Dans leur conclusion, Kaur et Kaur [KAU 16] ont insinué que les algorithmes AIS n'ont pas vraiment de bonnes performances sauf l'algorithme Immunos-1 quand il s'agit de logiciels critiques. Il s'avère très performant selon la mesure Rappel (Recall voir le chapitre 2 section 2.5).

Table 3. 1. Résumé des travaux sur les Système AIS dans la prédiction de défauts logiciels

Source	Algorithmes	Base de données	Métriques	Mesures de Performances	Résultat
CAT 07	AIRS 1	5 bases de données NASA	MCC MHAL LOC	G-mean1 G-mean 2 F-mesure	AIRS1 avec la méthode de sélection de caractéristiques CFS est plus performante dans le cas de large base de données.
CAT 07 A	AIRS1	Une seule base de données NASA	C&K	G-mean1 G-mean 2 F-mesure	AIRS 1 est plus performant que J48 (Arbre de décision) quand les 6 métriques C&K + LOC sont utilisées
CAT 09 B	AIRS1, AIRS 2.parallèle AIRS2, CLONALG, Immunos 1 et 2	5 bases de données NASA	OO MCC MHAL LOC	Area Under ROC Curve (AUC)	AIRS2. Parallèle est la meilleure méthode AIS quand les bases de données sont composées de métriques de niveau procédural alors qu'Immunos-2 est mieux dans le cas de mesures OO.
ABA 14	8 méthodes AIS	4 bases de données NASA	MCC MHAL LOC	AUC	Immunos-99 est meilleur quand une méthode de sélection de caractéristiques est employée, alors que c'est le contraire pour AIRS 2.parallèle.
KAU 16	8 méthodes AIS	3 bases de données extraites de 3 sous-projets Eclipse	MDP (Micro-interaction metrics)	Précision Rappel F-mesure	Immunos-1 est très bien adapté pour prédire des défauts de logiciels critiques.

Comme nous pouvons le constater à travers l'étude des travaux résumés précédemment ainsi que la table 3.1, il y a une très grande marge laissée pour étudier les systèmes immunitaires artificiels dans le domaine SFP.

En effet, tous les travaux ne s'intéressent qu'à un seul scénario SFP plus précisément l'intra-projets (CV) , laissant de côté l'inter-projet (PV) et le cross-projet (CPDP). De plus, ils utilisent presque le même ensemble de données fournies par la NASA avec des mesures de niveau procédural, alors que maintenant la programmation orientée objet est la plus utilisée dans le monde, donc des travaux utilisant les métriques OO devraient être le standard.

Dans ce chapitre, nous allons exposer notre vision au début de l'utilisation des méthodes d'inspiration immunologique dans le domaine SFP. La première partie du chapitre sera consacrée aux systèmes immunitaires dans SFP intra-projet utilisant des bases de données du référentiel PROMISE. La deuxième partie sera dédiée à notre proposition d'outil d'aide à la prédiction de défauts logiciels Cross-projet. Notre travail de thèse ne s'arrête pas ici puisque, dans le dernier chapitre, nous allons explorer les AIS dans un environnement inter-projets (PV).

3.2 Systèmes immunitaires artificiels pour la prédiction des défauts logiciels intra-projet (CV) :

Cette section va résumer le travail que nous avons effectué au début de cette thèse [HAO 17 A-HAO B] avec l'objectif d'évaluer les systèmes Immunitaires Artificiels (AIS) pour la prédiction de défauts logiciels (CV) utilisant une des bases de données ce trouvant dans référentiel de données PROMISE [MEN 16]. De plus, Nous allons faire une comparaison des résultats obtenus avec d'autres méthodes d'apprentissage automatique tel que les réseaux de neurones, forêt d'arbres décisionnels aléatoire (Random Forest) ...

Pour cela, nous avons mené une étude expérimentale de construction et d'évaluation de classifieurs pour SFP. Cette étude inclut douze algorithmes ML parmi eux 8 de type AIS, appliqués sur cinq bases différentes, issues de cinq logiciels orientés objet se trouvant dans le référentiel PROMISE spécialisé dans les données liées au domaine de génie logiciel. Les détails sur tous les algorithmes AIS utilisés dans notre démarche sont expliqués dans le chapitre 1 section 1.5.

3.2.1 Étude expérimentale effectuée

Cette section décrit les algorithmes d'apprentissage automatique, les bases données et les métriques que nous avons utilisés pendant notre étude expérimentale, ainsi que la méthode d'évaluation des performances des algorithmes sélectionnés pour la prédiction de défauts logiciels.

3.2.1.1 Méthodes d'apprentissage automatique

Les méthodes utilisées pour la construction des modèles de prédiction de défauts sont soit des méthodes issues des statistiques comme la régression logistique qui est l'une des

techniques les plus fréquemment implémentées, soit des méthodes d'apprentissage automatique (Machine Learning) pouvant être supervisées ou non supervisées, comme les réseaux de neurones, les arbres de décision ou le Clustering par les k-means [RAD 13-MAL 15-KAU 15].

Plusieurs algorithmes d'apprentissage ont été adoptés pour la prédiction de défauts logiciels. Selon plusieurs études [BEE 10-CAT 11-HAL 12-KAU 15-KUM 18]. Ces méthodes donnent de meilleurs résultats que les méthodes statistiques, considérées comme des méthodes à boîte noire où les relations entre les entrées et les sorties ne sont pas faciles à détecter ou analyser, de plus, elles sont très dépendantes des données.

Pour la construction de nos classifieurs, nous avons choisi l'outil WEKA (http://www.cs.waikato.ac.nz/ml/weka/). Les méthodes d'apprentissage que nous avons sélectionnées pour la composition de nos modèles de prédiction de défauts logiciels sont présentées en-dessous :

Les réseaux de neurones artificiels (MLP) : Le réseau neuronal artificiel le plus connu est le perceptron multicouches (Multi Layer perceptron ou MLP). C'est l'un des algorithmes les plus utilisées dans la prédiction. Une étude récente montre que le MLP est l'un des algorithmes les plus stables pour la prédiction de défauts logiciels [KAU 15].

Les arbres de décision (J48) : Plusieurs études ont montré que les arbres de décision s'adaptent très bien au problème de prédiction de défauts logiciels [JIA 08 -CAT 09 B-BEE 10-CAT 11-ABA 14]. Nous avons choisi d'utiliser J48 qui est une implémentation en java de l'algorithme C4.5.

Les forêts aléatoires d'arbres décisionnels (Random Forest RF) : Ce sont des classifieurs basés sur l'algorithme des arbres de décision. Les arbres sont composés sans élagage donc ils sont de très grande taille. Lorsque tous les arbres de la forêt sont construits, une nouvelle instance (classe de l'arbre) est fixée sur tous les arbres pour lancer un processus de vote pour la classification. Dans RF, la classe qui reçoit le plus de votes sera sélectionnée pour classer la nouvelle instance [JIA 08]. Selon la communauté des chercheurs dans le domaine de prédiction de défauts, RF est le meilleur algorithme d'apprentissage automatique recensé pour la prédiction de défauts [CAT 09 B -RAD 13-MAL 15].

Le Naïve Bayes (NB) : Selon Jiang et al [JIA 08] ainsi que He et al [HE 15], c'est l'un des algorithmes les plus simple mais les plus efficaces pour la prédiction de défauts.

3.2.1.2 Métriques choisies

Nous avons choisi les métriques orientées objet car ce sont les métriques les plus efficaces lors de la phase de pré-livraison des logiciels [RAD 13]. De plus, le paradigme orientées objet (OO) est le plus dominant à l'heure actuelle. En plus de la métrique LOC (nombre de ligne de code), la table 3.2 montre toutes les métriques utilisées dans notre travail. Pour plus d'information sur ces métriques, se référer au chapitre 2 section 2.4.

Table 3. 2. Métriques OO utilisées dans notre étude expérimentale

Les métriques	Source
WMC, CBO, NOC, DIT, LCOM, RFC	Chidamber et Kemerer,1994 [CHI 94]
LCOM3	Henderson-Sellers,1996 [HEN 96]
Ca, Ce	Martin ,1994 [MAR 94]
NPM, DAM, MOA, MFA, CAM	Bansiya et Davis, 2002 [BAN 02]
IC, CBM, AMC	Tang et al, 1999 [TAN 99]
Métriques de niveau procédural adaptées en OO dite Complexité cyclomatique (MCC) divisé en deux parties **MAX_CC**= MCC maximal d'une méthode qui se trouve dans la classe **AVG_CC**= moyenne CC de toutes les méthodes de la classe	McCabe,1976 [MCC 76]

3.2.1.3 Bases de données

Les valeurs des métriques choisies se trouvent dans les bases de données que nous avons utilisées. Nous avons sélectionné 5 bases différentes issues de 5 programmes orientés objet se trouvant dans le référentiel de données PROMISE[11]. Ces données sont de plus en plus utilisées dans le domaine, elles sont issues des travaux de Jureczko et Madeyski [JUR 10] ainsi que Jureczko et Spinellis [JUR 10 A] qui donnent plus d'information sur la manière avec laquelle les données sont rassemblées. Les logiciels open source d'où sont extraites ces bases de données, appartiennent à The Apache Software Foundation, ils sont tous écrits en langage Java. La table 3.3 présente un aperçu sur les bases de données utilisées**.** Les instances représentent les classes qui composent le programme.

Table 3. 3-Bases de données utilisées

Base de données	Nombre d'instances	Nombre d'instances fault prone	Taux % d'instances fault prone
Camel 1.6	965	188	19.50
Lucene 2.4	340	203	59.70
Poi 3.0	442	281	63.60
Xalan 2.5	803	387	48.20
Xerces 1.4	588	437	74.30

[11] http://openscience.us/repo/ Malheureusement, le site web n'est plus accessible. Cependant, les données peuvent être facilement obtenues avec une simple recherche Google ou dans le site personnel des auteurs : http://snow.iiar.pwr.wroc.pl:8080/MetricsRepo/

Avant d'utiliser ces données, nous avons effectué un pré-traitement, afin de les adapter à notre problème de classification binaire. A l'origine, La variable dépendante "bug" était représenté par des chiffres indiquant le nombre d'erreurs relevées dans la classe inspectée.

Pour remédier à cela, nous avons utilisé la même démarche effectué par certains chercheurs [HE 12-HE 15-ERT 15].en modifiant toutes les métriques bugs de toutes les instances des bases par ce qui suit :

- Si bug = 0, ceci indique qu'il n'y a pas de défauts, donc il est remplacé par "false" pour indiquer que la classe est not fault prone
- Si bug > 1, Alors elle remplacé par "true" pour dire que la classe est fault prone.

3.2.1.4 Méthode d'évaluation des performances

Pour l'évaluation des classifieurs dans le domaine de l'apprentissage automatique et la fouille de données, on se base souvent sur la matrice de confusion. C'est une matrice N x N où N est le nombre de classes, elle reporte comment le modèle a classé l'échantillon de test par rapport à sa vraie classe. Dans notre cas de prédiction de défauts logiciels, il s'agit d'un problème à deux classes (true : fault prone/false : not fault prone). Plus de détails et de justifications sur le choix de l'utilisation des mesures de performance sont expliqués dans la section 2.5 du chapitre 2.

	true	false
true	TP	FN
false	FP	TN

Matrice de confusion

Où :

- **TP** indique le nombre de vrais positifs qui représente le nombre d'exemples *true* qui sont classés *true*.
- **FN** : le nombre de faux négatifs qui représente le nombre d'exemples étiquetés *true* classés *false* par le modèle.
- **FP**: le nombre de faux positifs (faux rejet), exemples *false* classés *true*.
- **TN** : le nombre de vrais négatifs, qui représente le nombre d'exemples *false* classés comme tels par le modèle.

Plusieurs mesures d'évaluation des performances de l'apprentissage peuvent être calculées à partir de la matrice de confusion.

$$\text{Taux d'erreur} = \frac{FP+FN}{TP+FP+TN+FN}.$$

$$\text{Rappel (recall)} = \frac{TP}{TP+FN}.$$ (Sensibilité ou TVP : taux de vrais positifs)

Anti-Spécifité= 1-Spécificité $=1-\frac{TN}{TN+FP}=\frac{FP}{TN+FP}$ (taux de faux positifs FP)

Taux des exemples bien classés (puissance / accuracy) = $\frac{TP+TN}{TP+FP+TN+FN}$.

F-mesure (mesure F) = $\frac{2*(\text{Recall}*\text{Precision})}{(\text{Recall}+\text{Precision})}$

La méthode d'évaluation des performances d'apprentissage que nous avons utilisée est l'aire sous la courbe ROC (AUC-ROC). Dans le cas d'un classifieur binaire, il est possible de visualiser les performances du classifieur sur cette courbe. La courbe ROC est une représentation du taux de vrais positifs (recall) en fonction du taux de faux positifs (FP). Son intérêt est de s'affranchir de la taille des données de test dans le cas où les données sont déséquilibrées comme le montre la table 3.3 de nos bases d'apprentissage.

Cette représentation met en avant un nouvel indicateur qui est l'aire sous la courbe (AUC). Plus elle se rapproche de 1, plus le classifieur est performant. Selon plusieurs travaux, la puissance (accuracy), le rappel (recall) et la précision ne sont pas de bons indicateurs quand il s'agit de comparer des classifieurs pour la prédiction car les deux derniers ne prennent pas en compte le taux des faux positifs, ce qui conduit à une vue partielle de la classification [LES-CAT 12-ERT 15].

Lorsque la valeur de AUC est entre 0.5 et 0.7, elle dite faible, et si AUC est entre 0.7 et 0.9 la méthode est dite bonne pour certains type d'application et si c'est plus, on dit généralement, que la méthode a un très bon taux de prédiction [ZHO 10] . De plus, si AUC= 0.5 la classification effectuée est dite aléatoire.

3.2.2 Résultats obtenus

Dans cette section, nous allons fournier les résultats obtenus dans l'expérience décrite précédemment (section 3.2.1). Les paramètres de chacun des algorithmes sont par défaut. Pour les tests, nous avons utilisées la validation croisée (N Fold Cross Validation) où N=10, on divise l'échantillon original en N échantillons, puis on sélectionne un des N échantillons comme ensemble de test et les (N-1) autres échantillons constitueront l'ensemble d'apprentissage, la méthode sera répétée N fois. Par conséquent, toutes les données ont à la fois fait partie des ensembles d'apprentissage et ont été utilisées, au moins une fois, comme un ensemble de test. L'expérience est répétée 5 fois afin d'avoir des résultats plus significatifs, qui sont résumés dans la table 3.4 et la figure 3.1.

Table 3. 4-Résultats de l'expérience

Algorithme	Camel 1.6	Lucene 2.4	Poi 3.0	Xalan-2.5	Xerces-1.4	Total
AIRS1	0.561	0.602	0.696	0.564	0.683	**0.6212**
AIRS2	0.522	0.599	0.716	0.548	0.706	**0.6182**
AIRS2 Parallèle	0.55	*0.61*	0.713	0.593	0.663	**0.6258**
CLONALG	0.491	0.586	0.596	0.546	0.536	**0.551**
CSCA	*0.503*	0.603	*0.746*	*0.639*	*0.79*	**0.6562**

Immunos 1	0.566	0.57	0.565	0.549	0.757	**0.6014**
Immunos 2	0.5	0.5	0.5	0.588	0.5	**0.5176**
Immunos 99	0.531	0.558	0.559	0.548	0.747	**0.5886**
J48	0.616	0.687	0.772	0.677	0.913	**0.733**
NB	0.675	0.728	0.805	0.609	0.845	**0.7324**
MLP	0.706	0.733	0.773	0.689	0.901	**0.7604**
RF	**0.746**	**0.788**	**0.89**	**0.786**	**0.953**	**0.8326**

La figure 2 résume les résultats obtenus sur chacune des bases de données avec chacun des algorithmes d'apprentissage employés. D'après la table 3.4 et la figure 3.1, nous pouvons déduire quelques conclusions :

L'algorithme RF (forêts aléatoires d'arbres décisionnels) est le meilleur algorithme pour la prédiction de défauts logiciels parmi ceux que nous avons sélectionnés, ainsi les résultats obtenus sont en concordance avec ceux trouvés dans la littérature [CAT 09 B- ABA 14].

Une nouveauté que nous avons obtenue, est que le classifieur CSCA est le meilleur classifieur immunitaire (AIS) pour la prédiction de défauts logiciels, ce qui va à l'encontre des travaux de Abaei et Selamat [ABA 14], les seuls ayant utilisé CSCA pour la prédiction de défauts. D'autre part, l'implémentation parallèle de l'algorithme AIRS (AIRS2 parallèle) donne des résultats proches de ceux d'AIRS 1 et 2, aux alentours de 0.62 et non meilleurs comme dans [CAT 09 B, ABA 14].

Les implémentations de l'algorithme Immunos-81 (Immunos 1-2 et 99) ne donnent pas des résultats pas très concluants, surtout Immunos 1 et 2 dont on ne peut pas modifier les paramètres pour essayer d'adapter l'algorithme car ils ne sont pas disponibles [BRO 05 B] comme CLONALG et Immunos-99.

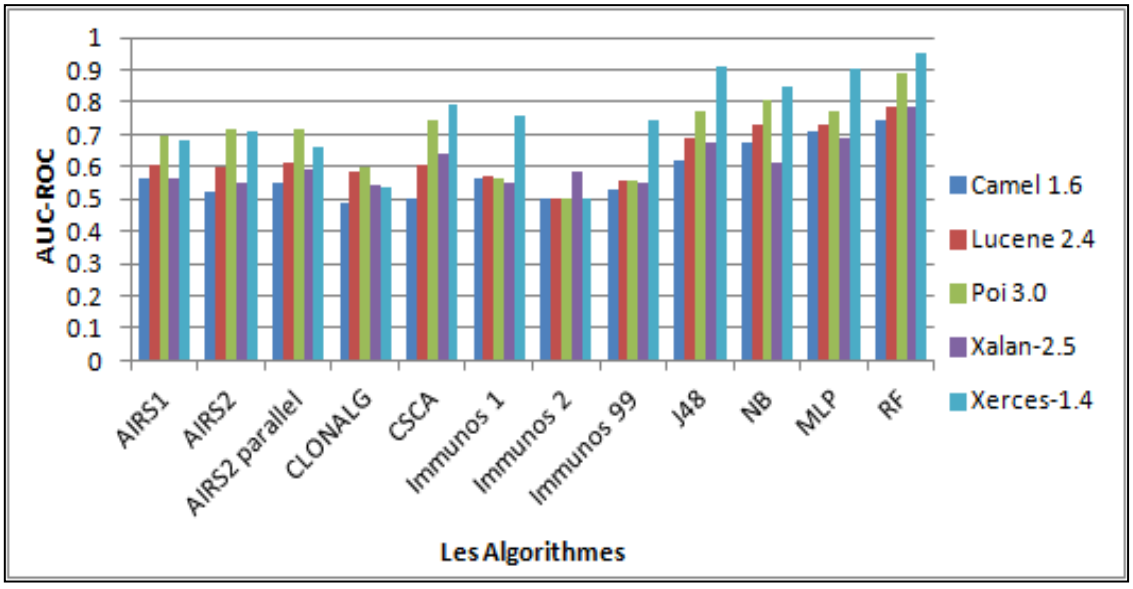


Figure 3. 1- Les valeurs AUC des expérimentations sur les 5 bases de données

Pour voir si nos conclusions sont exactes, nous avons étendu notre investigation sur les performances des AIS dans l'environnement SFP intra-projets. Pour cela, nous avons reconduit la même expérience. Mais cette fois, au lieu de 5 bases de données, nous avons ajoutés 5 autres de la même famille de référentiel de données PROMISE représentant 10 projets open source collectés par [JUR 10-JUR 10 A]. Le résultat de cet effort est présenté dans la table 3.5 et la figure 3.2.

Table 3. 5- Résultats de l'expérience étendue

Algorithmes	Camel 1.6	Lucene 2.4	Poi 3.0	Xalan 2.5	Xerces 1.4	Ant 1.6	Ivy 1.4	Jedit 4.0	Synapse 1.2	Velocity 1.5	TOTAL
AIRS1	0.561	0.602	0.696	0.564	0.683	0.675	0.491	0.609	0.644	0.578	0.6103
AIRS2	0.522	0.599	0.716	0.548	0.706	0.661	0.522	0.604	0.633	0.62	0.6131
AIRS2 par	0.55	**0.61**	0.713	0.593	0.663	0.647	0.488	0.535	**0.668**	0.598	0.6065
CLONALG	0.491	0.586	0.596	0.546	0.536	0.645	0.482	0.477	0.604	0.626	0.5589
CSCA	0.503	0.603	**0.746**	**0.639**	**0.79**	**0.719**	0.558	0.601	0.65	0.644	**0.645**
Immunos 1	**0.566**	0.57	0.565	0.549	0.757	0.603	**0.622**	**0.613**	0.571	**0.668**	0.6084
Immunos 2	0.5	0.5	0.5	0.588	0.5	0.568	0.5	0.5	0.618	0.5	0.5274
Immunos 99	0.531	0.558	0.559	0.548	0.747	0.621	0.488	0.612	0.565	0.572	0.5801

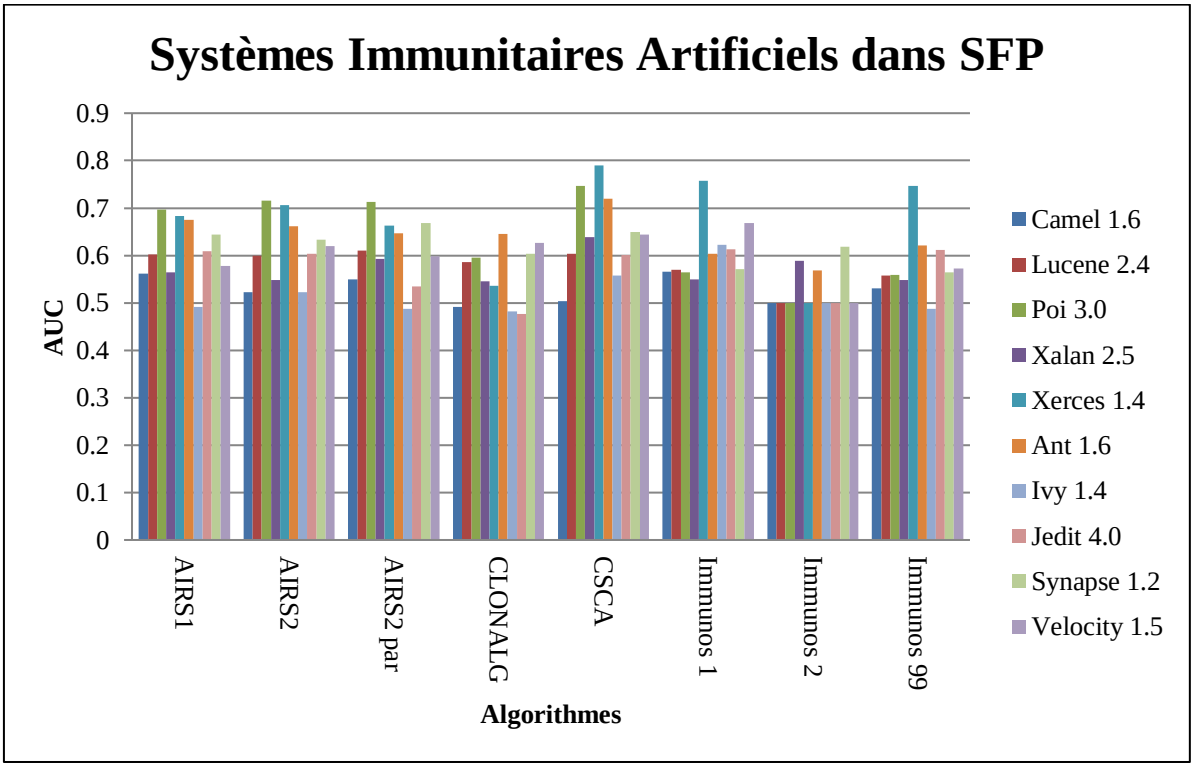


Figure 3. 2-Les valeurs AUC de l'expérience étendue sur les 10 bases de données

Même l'expérience étendue n'a rien changé à notre première conclusion, où la meilleure méthode AIS reste CSCA. Par contre, le temps d'apprentissage de cette méthode est assez long surtout en comparaison aux autres systèmes immunologiques. L'autre observation nous

conforte sur le fait qu'Immunos-2 n'est pas vraiment adapté à ce genre d'étude bien que les métriques employées soient toutes orientée objets.

3.2.3 Discussion

Nous avons essayé, dans ce travail, à travers notre étude expérimentale, de déterminer le meilleur algorithme inspiré des systèmes immunitaires artificiels pour prédire les défauts, sur des bases de données autres que celles utilisées dans les rares travaux évaluant les AIS dans ce domaine. Nous avons aussi comparé les résultats obtenus avec ceux d'autres algorithmes d'apprentissage automatique issus de la littérature. Ce travail a été présenté à deux conférences internationales [HAO 17 A-HAO 17 B].

Les résultats de nos expérimentations montrent que le meilleur algorithme parmi notre sélection est RF (Random Forest), tandis que la meilleure méthode issue des systèmes immunitaires artificiels est (CSCA Clonal Selection Classifier Algorithm) malgré un temps d'exécution assez long, surtout comparée avec les autres AIS. Nous avons pu voir qu'Immunos 1 et 2 ne sont pas très adaptés pour résoudre ce problème de prédiction, surtout dans le cas d'Immunos 2 qui se comporte, 4 fois sur 5, comme un algorithme de classification aléatoire d'après sa valeur ROC-AUC (AUC= 0.5). Nous avons étendu notre expérimentation à 10 bases de données pour généraliser notre conclusion. Nos observations sont restées les mêmes quand il s'agit de la prédiction de défauts intra-projets.

Dans la suite de ce chapitre, nous allons exposer notre proposition d'implémentation d'un outil d'aide à la prédiction de défauts logiciels dans un scénario Cross-projets (CPDP). Le fruit de ce travail a conduit à une présentation orale dans une conférence internationale [HAO 17].

3.3 Outil d'aide à la prédiction de défauts logiciels

Pour rappel, les études de prédiction de défauts logiciels sont séparées en trois grands scénarios. Tout dépend des données d'apprentissage (pour construire le modèle) et les instances de test (le logiciels cible/ la partie à vérifier). Ces schémas sont la prédiction de défauts Intra (CV), Inter (PV) et Cross-projets (CPDP). Pour plus de détails sur les différences entre ces scénarios, il faut revenir à la section 2.3.3 du chapitre 2.

Durant la dernière décennie, les travaux étudiant le Cross-projets (CPDP) connaissent un intérêt grandissant dans la communauté SFP. La perspective de construire des modèles de prédiction à partir de données qui n'ont pas de lien avec le projet en développement, sans passer par l'effort laborieux de calcul et maintenance de ce genre de base de données, est très attirante [RAT 17-KHA 21]. Pour simplifier, les données d'apprentissage d'un modèle de prédiction CPDP sont extraites d'un logiciel ou projet complètement différent du système dont nous voulons vérifier les modules (qui sont enclins à détenir des défauts ou non), afin d'établir un programme de test raffiné, réduisant par la même occasion le temps et le budget de cette phase de développement très gourmande [ZIM 09-HE 12-RAD 13-RAT 17-KUM 18-KHA 21].

Malgré un nombre important de travaux de recherche dans le domaine de la prédiction de défauts logiciels, ce procédé n'est pas encore couramment appliqué dans l'industrie logicielle. Ceci est principalement dû au fait que les travaux publiés ne se concentrent que sur l'efficacité prédictive des méthodes édifiées, sans prendre en considération les besoins réels de l'industrie logicielle [RAN 14]. Par conséquent, on constate un manque accru d'outils permettant de faciliter le travail d'un chef de projet afin de prédire les défauts logiciels avant leur apparition [LES 08]. Ceci est particulièrement le cas pour les nouveaux projets pour lesquels on ne dispose pas de données antérieures permettant de simplifier le processus de prédiction.

Dans cette optique, l'objectif de notre travail est de proposer un outil d'aide à la prédiction de défauts logiciels Cross-projets. Afin d'avoir une idée précise des caractéristiques que notre outil devrait avoir pour être utile et efficace, nous avons mené une étude expérimentale d'édification et d'évaluation de classifieurs pour la prédiction de défauts logiciels. Cette étude inclut six algorithmes d'apprentissage automatique appliqués sur cinq bases différentes, issues de cinq logiciels orientés objet se trouvant dans le référentiel PROMISE [MEN 16] spécialisé dans les données liées au domaine du génie logiciel.

3.3.1 Étude expérimentale effectuée

Cette section décrit les algorithmes d'apprentissage automatique, les bases données et les métriques que nous avons utilisés pendant notre étude expérimentale, ainsi que la méthode d'évaluation des performances des algorithmes sélectionnés pour la prédiction de défauts logiciels.

Pour éviter les répétitions et la surcharge du chapitre, la table 3.6 résume les acteurs principaux utilisés dans cette expérimentation. Car, nous avons employée presque toutes les méthodes, bases de données, métriques logicielles et mesures de performances déjà exposée dans l'étude précédente (Voir la section 3.2.1).

Comment nous l'avons expliqué, le but de cette section est de présenter une proposition d'un outil d'aide à la prédiction de défauts logiciels, même si notre objectif principal est plutôt l'étude de l'apport des systèmes immunitaires dans la prédiction de défauts logiciels. Dans cette partie de notre travail [HAO 17], présentée dans cette section, nous nous sommes concentrés beaucoup plus sur l'implémentation de l'outil que sur l'étude de l'apport des systèmes immunitaires.

Table 3. 6- Résumé des méthodes, métriques, bases de données et mesures de performance utilisées

Algorithmes	Base de données	Métriques	Mesure de performance
AIRS 1 AIRS 2 NB MLP J48 (C 4.5) RF	Nous avons utilisé 5 bases de données du référentiel PROMISE (aller à la Table 3.3 pour plus de détails)	les bases de données employées contiennent des instances composés de 19 métriques orientées objets + LOC (voit la Table 3.2)	Afin d'évaluer les performances des classifieurs appliquées dans cette étude nous avons utilisés la mesure AUC (area under the ROC curve). Pour plus de détaille aller à la section 3.2.1.4 .

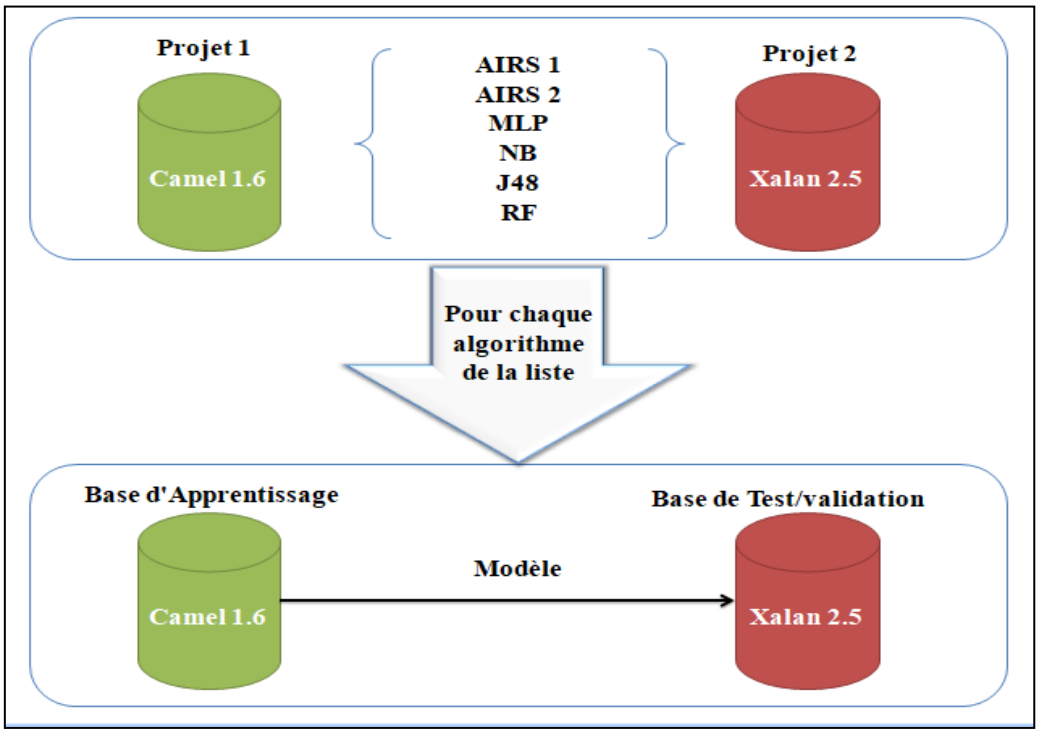


Figure 3. 3- Exemple de validation CPDP utilisé

Comme le titre du chapitre va inciter le lecteur à s'attendre que la recherche conduite ici se concentre sur les systèmes immunitaires artificiels, nous avons choisi deux algorithmes AIS, plus précisément deux versions de la méthode AIRS (Artificial Immune Recognition System) (voir la section 1.5.1 du chapitre 1) parmi les techniques de Machine Learning appliquées dans cette expérimentation.

Système immunitaire artificiel pour la reconnaissance (AIRS) : Artificial Immune recognition System (AIRS) a été introduit par [WAT 01]. C'est un algorithme d'apprentissage automatique supervisé, nécessitant l'utilisation d'un ensemble d'antigènes comme données d'apprentissage dont le système doit produire un ensemble d'anticorps utiles pour la phase de classification. En plus de la première version AIRS1, une deuxième, nommée AIRS2, a été proposée comme résultat de la collaboration de Watkins, Timmis et Boggess [WAT 04]. AIRS2 présente quelques différences mineures par rapport à AIRS1, mais est moins complexe que son prédécesseur. À la fin de la phase d'apprentissage des deux versions de l'algorithme AIRS, nous obtenons un ensemble de cellules mémoires permettant d'effectuer une classification et cela grâce à la méthode des k-plus proche voisin (kppv). AIRS a été utilisé avec succès dans plusieurs problèmes de classification [BRO 05] et a été aussi appliqué dans le domaine de la prédiction de défauts logiciels (Table 3.1).

Pour simuler le scénario CPDP (Figure 3.3), nous avons choisi la simple validation par paire (voir la section 2.3.3 du chapitre 2), ci-dessous une simplification des étape suivies pour effectuer notre analyse :

- Phase 1 : les données sont divisées en deux parties : une des bases (par exemple Camel 1.6) va être la base d'apprentissage, alors que les 4 autres, une à une, vont servir comme base de test.
- Phase 2 : nous sélectionnons une des autres bases qui n'a pas servi comme base d'apprentissage puis répétons la première phase.
- L'expérience est conduite en utilisant les algorithmes d'apprentissages automatiques cités dans la Table 3.6.
- L'expérience est répétée 5 fois afin d'avoir des résultats plus fiables.

3.3.2 Résultats obtenus

Les résultats de notre étude expérimentale relative à la construction et l'évaluation des classifieurs vont nous servir de base pour configurer notre outil d'aide à la prédiction de défauts logiciels. La table 3.7 suivante, montre les résultats que nous avons obtenus lors des expérimentations décrites dans la section précédente, ils sont classés par base d'apprentissage, et indiquent la moyenne ROC-AUC/puissance obtenue avec chaque algorithme en utilisant les autres bases comme échantillon de test des modèles construits. Une synthèse de ces résultats est fournie par la Table 3.8 et la Figure 3.4.

Dans les Tables 3.7 et 3.8, si les valeurs de la mesure de performance sont en rouge et italique, cela veut dire qu'une des méthodes AIS a été plus performante que l'autre. Mais si le résultat est affiché en gras, alors c'est la plus haute performance pour la paire (données d'apprentissage/ données de test) obtenue parmi tous les algorithmes sélectionnés

Table 3. 7- Résultats de l'expérimentation Cross-Projets

base d'apprentissage	***Camel 1.6***			
bases de test	**Lucene 2.4**	**Poi 3.0**	**Xalan 2.5**	**Xerces 1.4**
AIRS 1	*0.528/47.94*	*0.545/49.09*	*0.513/52.30*	*0.594/43.53*
AIRS 2	0.489/43.23	0.537/45.70	0.504/51.55	0.493/34.01
J48	0.558/54.11	0.575/59.95	0.573/55.66	0.553/38.77
NB	0.621/52.05	0.548/47.05	0.543/54.91	0.531/37.75
MLP	0.535/43.23	0.664/57.23	0.515/55.79	0.506/39.62
RF	**0.66/45.29**	**0.708/42.30**	**0.581/54.67**	**0.718/32.65**
base d'apprentissage	***Lucene 2.4***			
bases de test	**Camel 1.6**	**Poi 3.0**	**Xalan 2.5**	**Xerces 1.4**
AIRS 1	*0.534/53.78*	0.528/49.32	0.536/53.3	*0.694/59.69*
AIRS 2	0.522/49.32	*0.529/48.86*	*0.545/54.17*	0.652/58.84
J48	0.593/54.19	**0.824/78.95**	0.538/54.54	0.939/95.40

NB	0.626/70.88	0.806/61.99	**0.595/57.40**	0.854/66.15
MLP	0.616/53.47	0.682/62.66	0.547/54.04	0.924/94.89
RF	**0.64/47.25**	0.788/77.14	0.59/53.05	**0.999/99.65**
base d'apprentissage	***Poi 3.0***			
bases de test	**Camel 1.6**	**Lucene 2.4**	**Xalan 2.5**	**Xerces 1.4**
AIRS 1	0.54/60.41	0.559/55.88	*0.527/53.05*	0.595/46.59
AIRS 2	*0.57/63.21*	*0.626/61.47*	0.526/52.92	*0.615/46.93*
J48	0.53/48.29	0.687/67.64	0.578/57.53	0.485/53.06
NB	0.612/74.09	0.672/55.29	0.561/54.91	0.666/40.13
MLP	0.547/63.93	0.687/60	0.581/56.03	**0.705/50.17**
RF	**0.621/55.85**	**0.705/65**	**0.586/54.54**	0.663/51.53
base d'apprentissage	***Xalan 2.5***			
bases de test	**Camel 1.6**	**Lucene 2.4**	**Poi 3.0**	**Xerces 1.4**
AIRS 1	*0.577/60.10*	*0.622/59.70*	*0.555/50.67*	0.603/44.89
AIRS 2	0.537/67.66	0.582/55.58	0.519/45.24	*0.625/51.70*
J48	0.535/51.60	0.552/54.41	0.578/61.08	0.709/60.20
NB	**0.635/78.03**	**0.705/48.52**	**0.72/44.11**	**0.713/36.05**
MLP	0.532/63.52	0.497/48.23	0.478/44.79	0.494/43.36
RF	0.574//65.59	0.572/55.29	0.681/54.52	0.668/48.80
base d'apprentissage	***Xerces 1.4***			
base de test	**Camel 1.6**	**Lucene 2.4**	**Poi 3.0**	**Xalan 2.5**
AIRS 1	0.485/32.64	0.567/62.05	*0.586/66.06*	0.492/48.19
AIRS 2	*0.564/35.33*	*0.572/63.82*	0.578/68.09	*0.526/51.30*
J48	0.561/26.21	0.536/59.11	0.558/64.93	0.507/48.06
NB	0.579/46.52	0.698/70	**0.796/76.24**	**0.569/53.54**
MLP	**0.629/26.63**	**0.684/60.29**	0.705/65.38	0.532/46.94
RF	0.568/23.73	0.634/59.41	0.782/64.93	0.544/48.69

Une première observation, nous montre que les deux algorithmes AIS choisis ont presque les mêmes performances que certaines méthodes d'apprentissage automatique connues telles que J48 (les arbres de décision). Si nous ne nous intéressons qu'aux méthodes AIS, la version 1 de la technique AIRS, dans la majorité des cas, a un petit avantage sur AIRS 2. Par contre, leurs résultats restent très faibles et proches d'un classifieur aléatoire d'après la mesure de performance AUC.

Notons que le modèle édifié par l'algorithme AIRS 1 et la base de données Xalan 2.5 est meilleur que le classifieur RF pour prédire les défauts du système Camel 1.6.

Ce dernier algorithme a les meilleurs résultats d'après notre expérimentation, sur 3 des 5 bases d'apprentissages sélectionnés (en gras dans les tables 3.7 et 3.8), sauf dans le cas où les instances des logiciels Xalan 2.5 et Xerces 1.4 sont employées pour édifier les modèles CPDP.

Table 3. 8- Synthèse de des résultats obtenus

Algorithme	Camel 1.6	Lucene 2.4	Poi 3.0	Xalan 2.5	Xerces 1.4
AIRS 1	*0.545*	*0.573*	0.55525	*0.58925*	0.5325
AIRS 2	0.50575	0.562	*0.58425*	0.56575	*0.56*
J48	0.56475	0.7235	0.57	0.5935	0.5405
NB	0.56075	0.72025	0.62775	**0.69325**	**0.6605**
MLP	0.555	0.69225	0.63	0.50025	0.6375
RF	**0.66675**	**0.75425**	**0.64375**	0.62375	0.632

Pour résumer, le meilleur algorithme pour la prédiction Cross-projet est RF (Random Forest), ce qui concorde avec la littérature du domaine où RF est considéré comme un des meilleurs algorithmes d'apprentissage automatique pour la prédiction de défauts logiciels. De plus, dans notre cas la base Lucene 2.4 est la base qui a permis d'avoir les meilleures performances par tous les algorithmes que nous avons utilisés dans notre étude. Par exemple, si nous prenons Lucene 2.4 comme base d'apprentissage pour l'algorithme RF, cela nous a permis d'avoir 99.65 % de taux de classification des instances de la base Xerces 1.4, ce qui nous a encouragé à utiliser RF comme méthode d'apprentissage et que Lucene 2.4 sera la base d'apprentissage par défaut pour notre outil.

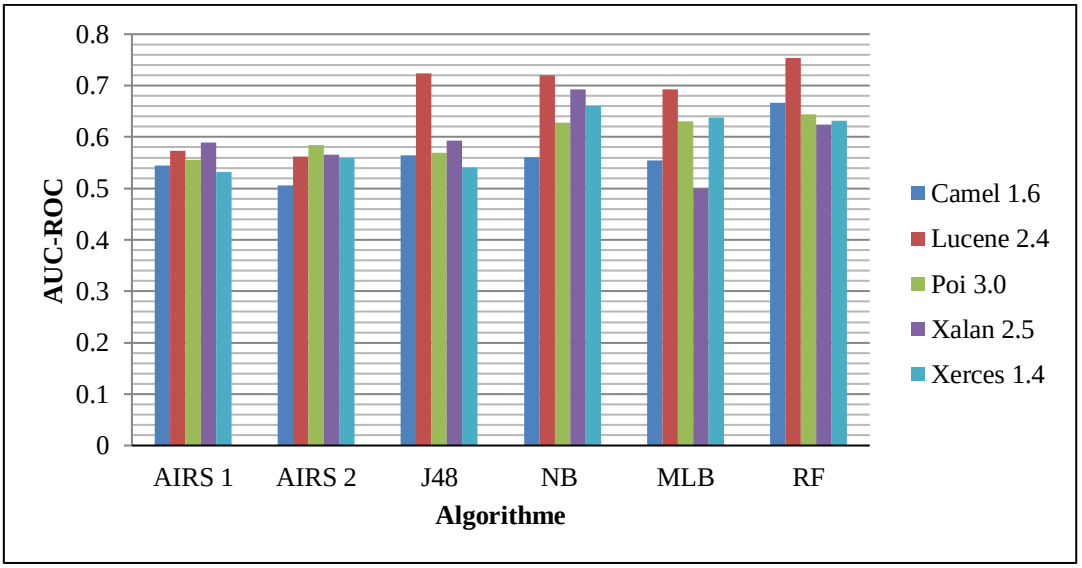


Figure 3. 4-Aperçu des résultats obtenus

Sur la base des résultats obtenus, nous avons procédé à l'implémentation d'un outil qui a pour but d'aider les praticiens à intégrer la prédiction de défauts logiciels dans leur cycle de développement.

3.3.3 Présentation de l'outil réalisé

Malgré un nombre assez élevé de travaux sur la prédiction de défauts logiciels, il y a un manque notable d'outils dédiés précisément à cet effet. Il y a certes beaucoup de logiciels capables de calculer les métriques, qu'elles soient orientées objet ou autres, mais ces logiciels sont majoritairement propriétaires et payants. Par contre, plusieurs outils proposent en open source des algorithmes d'apprentissage automatique facilement accessibles, tels que WEKA.

Les quelques rares propositions d'outils automatiques de prédiction de défauts logiciels se focalisent sur la prédiction CV, comme l'outil de Ostrand et Weyuker [OST 10] qui utilise l'algorithme de régression binomiale négative, ainsi que le Plugin Eclipse implémenté par Catal et al [CAT 11 A] qui utilise l'algorithme Naïve Bayes (NB).

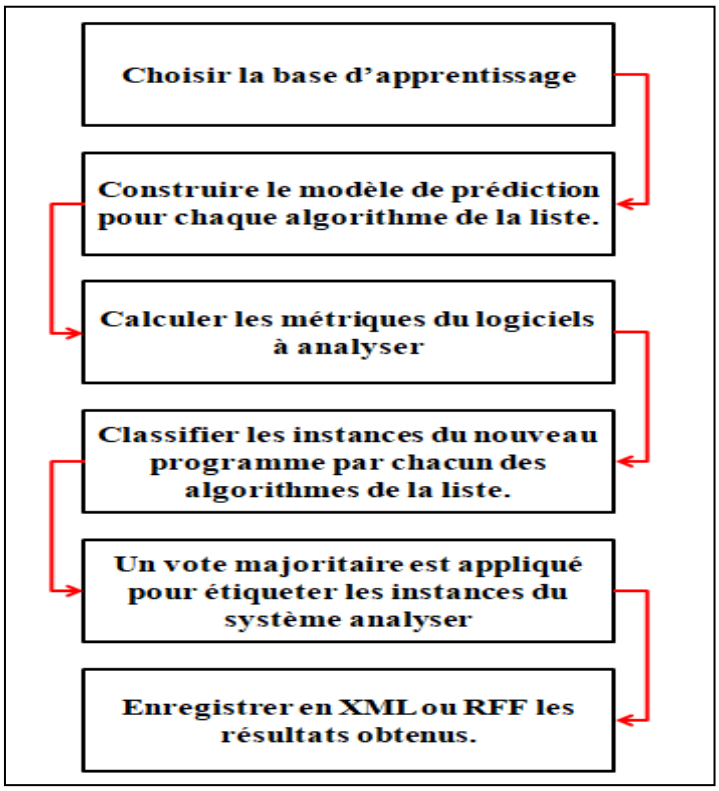


Figure 3. 5-Simplification du fonctionnement de notre outil

Dans la suite de cette section, nous allons présenter notre propre outil d'aide à la prédiction de défauts logiciels. Cet outil doit prendre en compte tous les résultats trouvés dans l'étude expérimentale effectuée afin de mieux cibler les modules sujets aux défauts lors de la phase de test. Il doit être capable de calculer les métriques orientés objets, et doit utiliser un des algorithmes de prédiction.

Une des raisons pour lesquelles il y a un manque d'outils capables de prédire les défauts logiciels est lié au coût de développement de ce genre d'applications. Pour cette raison, nous avons utilisé dans notre outil des composants open source et gratuits, et qui sont :

- **CKJM (Chidamber and Kemerer Java Metrics)[12] [SPI 05]:** Ce logiciel nous a permis de construire les bases de données que nous avons utilisées lors de nos expérimentations. Cependant, nous avons adapté cette API afin de pouvoir calculer les métriques des programmes écrits en langage java.
- **WEKA**, représentent le cœur de notre outil. Contenant la majorité des méthodes d'apprentissage automatique utilisées dans notre expérimentation. Pour plus de détails aller au chapitre 2 section 2.9
- **Langage XML :** Pour des raisons de portabilité le langage XML est employé pour stocker à la fois les métriques des logiciels analysés, ainsi que les résultats produits par notre outil.

D'après l'étude que nous avons menée, le taux de classification est très faible, même en utilisant la meilleure configuration recensée dans notre expérience, comme Lucene 2.4 + RF cela n'a donné que 53.05 % de réussite sur la base Xalan 2.5, ce qui n'encouragera pas les praticiens à employer la prédiction de défauts Cross-projet dans leur cycle de vie de développement. Pour remédier à cela, nous proposons l'algorithme suivant (figure 3.5) :

Début

listMethode: la liste de méthode d'apprentissage ;

tableVote: tableau de type String pour le vote;

dataSet: base d'apprentissage;

listInstance: liste d'instances à classer

listResutats: liste pour le résultats final;

Pour chaque *al* de *ListMethode* :

1- *al. apprendre(dataSet)* ;

fin;

Pour chaque *i* de *ListInstance* :

1- *resultat* ← *al .classe(i);*

2- *tableVote* ← *resutats* ;

3- *répéter 1 et 2 pour tout al de ListMethode*;

4 - *listeResutats* ← *appliquer.Vote(tableVote);*

fin

Fin

[12] https://www.spinellis.gr/sw/ckjm/

Concernant les méthodes d'apprentissage des classifieurs intégrées dans l'outil proposé, nous avons retenu les trois meilleurs algorithmes qui nous ont permis d'obtenir les meilleurs résultats pendant notre étude expérimentale, et qui sont respectivement RF, NB et MLP. Nous avons choisi d'appliquer le principe de vote majoritaire entre ces trois classifieurs pour classer une nouvelle entité par notre outil.

En ce qui concerne l'interface graphique de notre outil, elle est simple et conviviale et intègre toutes les fonctions dont un développeur a besoin pour incorporer la prédiction de défauts dans le cycle de vie d'un logiciel développé ; plus précisément, avant la phase de test, et lors de la factorisation dans la phase de maintenance.

Notre outil fournit deux modes d'utilisation : *normal* et *avancé*. Le premier mode (figure 3.6) est assez simple. Il faut juste importer le dossier qui contient le byte code (.class) de l'application que l'utilisateur veut analyser, et la base d'apprentissage sera directement intégrée (aucune connaissance supplémentaire n'est requise). Notons que la base d'apprentissage par défaut est Lucene 2.4.

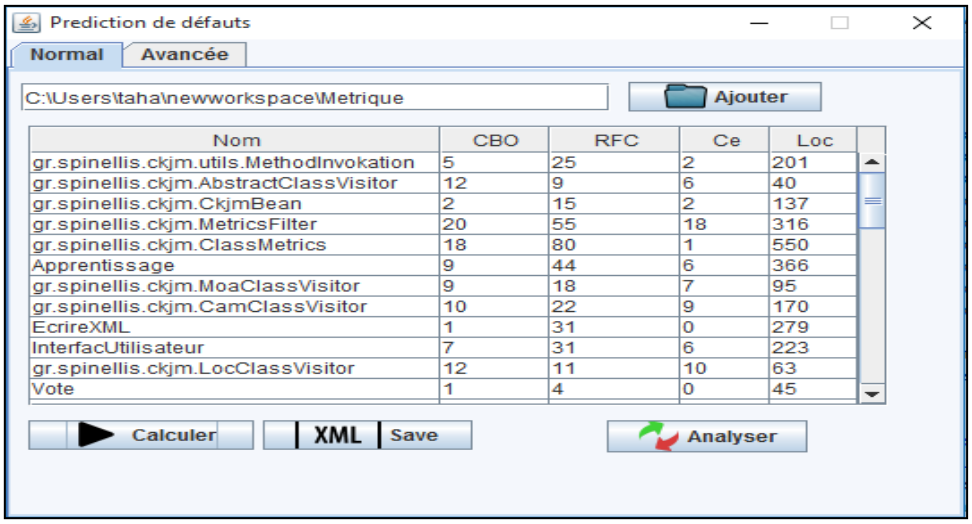


Figure 3. 6-Mode normal de l'outil d'aide proposé

Le deuxième mode d'utilisation (figure 3.7) est aussi simple que le premier, mais donne l'avantage à l'utilisateur d'utiliser sa propre base d'apprentissage. Le fichier doit être de type arff. Bien entendu, une certaine connaissance est requise pour manipuler ce genre de fichier ; ce qui facilite l'utilisation du plug-in Weka lors de l'apprentissage. En appuyant sur le bouton point d'exclamation, une aide est fournie pour permettre de bien écrire l'en-tête de fichier arff manipulé pour éviter les problèmes de classification.

Pour avoir les résultats de l'analyse, c'est à dire la classification (figure 3.8), il suffit juste d'appuyer sur le bouton *analyser* dans n'importe quel mode d'utilisation. Rappelons que le terme *fault prone* désigne le fait que cette classe contienne des défauts ou pas (*true* pour oui, *false* pour non). Cette interface permet de sauvegarder les résultats soit en format XML, soit en format de fichier arff pour une future utilisation.

En appliquant l'outil sur les bases de données utilisés lors de notre expérimentation dans le mode normal, le taux de classification à largement augmenté, là où la meilleur configuration

RF+ Lucene 2.4 avait un taux 53.05% de classification sur les instances du programme Xalan-2.5, l'outil est arrivé à produire le résultat de 76.5 %. Un seul cas négative a pu être observé, c'est quand la base de test est Xerce-1.4 où le taux s'est réduit de 99.65 avec la meilleur configuration à 95%, Ce qui reste acceptable à notre avis.

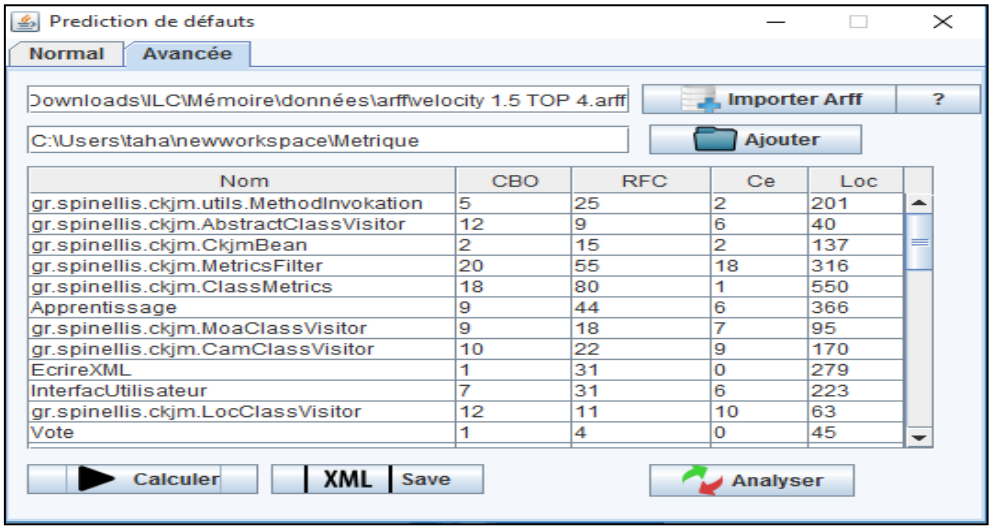


Figure 3. 7-Mode avancé de l'outil d'aide proposé

3.3.4 Discussion

Dans cette section, nous avons exposé notre proposition d'implémentation d'un outil d'aide à la prédiction de défauts logiciels Cross-projets (CPDP). Notre proposition se base sur une étude expérimentale que nous avons menée afin de construire et d'évaluer 6 classifieurs pour la prédiction de défauts logiciels CPDP.

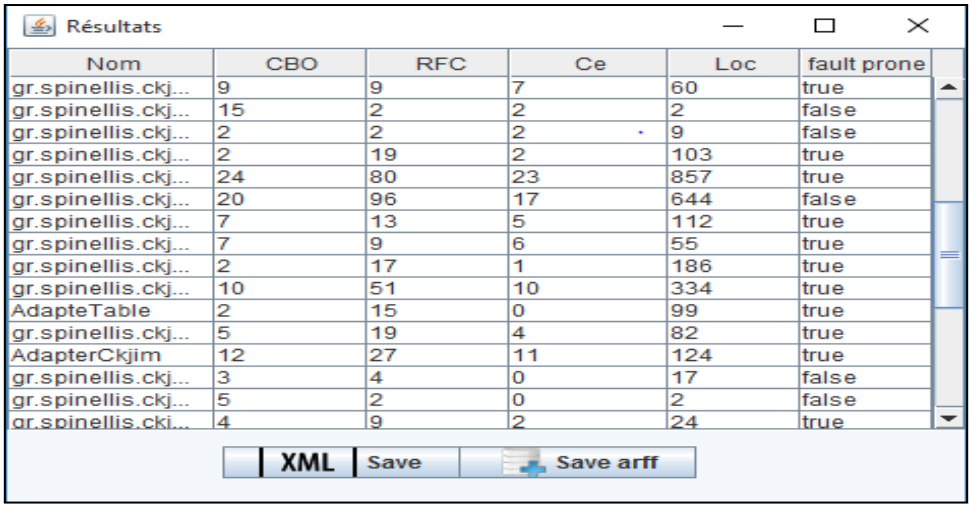


Figure 3. 8-Résultats d'analyse fournis par l'outil proposé

Ces algorithmes sont : les systèmes immunitaires AIRS1 et AIRS2, le Perceptron Multicouches (MLP), les Arbres de Décision (j48), Random Forest (RF) et naïve bayes (NB).

L’évaluation a été effectuée sur cinq bases de données différentes issues du référentiel de données PROMISE [MEN 16] spécialisé dans les données liées au domaine de génie logiciel.

Les résultats que nous avons obtenus, suite à notre étude, nous ont permis de confirmer la faisabilité de la prédiction CPDP mais aussi sa difficulté relative aux choix des données et des méthodes d’apprentissage pour la construction des modèles de prédiction.

Concernant notre outil, il permet d’aider à mieux exploiter les ressources disponibles, surtout pendant la phase de test, pour améliorer la qualité des logiciels développés et d’en réduire les coûts. Des essais sur la prédictivité de notre système ont montré une amélioration des taux de classification, ce qui nous confirme l'utilité d'un tel programme.

Actuellement, cet outil ne fonctionne que sur des programmes écrits en langage java, et présente encore quelques problèmes de performances. Nous travaillons sur une nouvelle version pour prendre en compte plusieurs améliorations telles que :

- Collecter des instances dans des bases qui contiennent des données assez équilibrées entre les deux classes fault prone et not fault prone, comme la base Lucene 2.4, par exemple.
- Choisir un algorithme d'apprentissage assez stable comme RF ou MLP [KAU 15]
- Etendre l'approche utilisée afin de prendre en compte des programmes écrits en des langages autres que Java.
- Augmenter la prédictibilité des classifieurs intégrés dans l’outil en remplaçant, au fur et à mesure, les données de la base d'apprentissage par des données issues d'autres réalisations. Ce qui revient à orienter cet outil vers la prédiction de défauts Inter-projets (PV).
- Permettre au logiciel d'apprendre de plusieurs sources de données.

3.4 Conclusion

Dans ce chapitre, nous avons commencé à étudier l’apport des systèmes immunitaires artificiels (AIS) dans la prédiction de défauts logiciels (SFP). Nous avons conduit trois études qui ont abouti à trois articles présentés dans des conférences internationales [HAO 17-HAO 17 A-HAO 17 B].

Dans la première partie du chapitre, nous avons présenté l'évaluation des systèmes AIS dans SFP intra-projets (voir la section 2.3.3 du chapitre 2) [HAO 17 A-HAO 17 B]. Pour cela, nous avons conduit une expérimentation où 8 algorithmes AIS (voir la section 1.5 du chapitre 1) ont aidé à construire des modèles de prédictions de défauts logiciels intra-projets (CV) sur 5 bases de données extraites du référentiel PROMISE [MEN 16]. Les instances de ces bases sont composés principalement de métriques logicielles orientées objets. Les résultats obtenus sont comparés avec des modèles construits avec des algorithmes d'apprentissage référencés dans la communauté SFP tels que les réseaux de neurones (MLP), les arbres de décision(J48), Forêt d'arbres décisionnels (RF) ...

Notre évaluation a montré que la méthode RF est la meilleure parmi notre sélection d'algorithmes d'apprentissage, pour la prédiction de défauts CV. La surprise qui nous attendait est que la méthode AIS la plus adaptée soit l'algorithme CSCA, cette technique hybride a surpassé les performances de l'algorithme AIRS 2.paralèlle réputé comme le meilleur système immunologique dans la littérature SFP. Cependant, les performances de la méthode Immunos-2 sont semblables à celles d'un classifieur aléatoire, où ses résultats dépassent rarement les 50% d'après la mesure de performance AUC (Area Under the ROC Curve).

La deuxième partie du chapitre a, quant à elle, exposé notre tentative d'implémentation d'un outil d'aide à la prédiction de défauts Cross-projets (CPDP) [HAO 17]. Dans notre démarche, nous avons mené une expérience afin de mieux configurer notre système. Dans cette démarche nous avons utilisés les mêmes algorithmes, bases de données, métriques logicielles et mesures de performance que la première expérimentation. Bien entendu, quelque différence s'imposait, où seuls deux systèmes AIS ont été utilisés dans cette étude qui sont les deux versions de l'algorithme AIRS. Un autre changement est bien lié à la nature du scénario dans lequel se trouve notre étude. Le scénario CPDP impose que les bases d'apprentissage et test soient extraites de deux logiciels différents (pour plus de détails aller au chapitre 2 section 2.3.3).

Après avoir analysé les résultats de notre expérience, nous avons pu implémenter un système assez simple pour aider les praticiens à introduire la prédiction de défauts logiciels dans leur cycle de développement en n'utilisant que des API open source afin de réduire le coût que pourrait engendrer l'implémentation d'un tel programme. Malgré la simplicité du système, il est arrivé à améliorer le taux de classification grâce à l'incorporation d'un système de vote pour l'étiquetage des instances du système analysé.

Comme vous pouvez le voir, les résultats de ce chapitre sont assez variés, surtout par rapport au scénario SFP étudié. Par contre, il manque encore un autre schéma, qui est la prédiction de défauts Inter-projet (PV). Nous allons consacrer le prochain chapitre à la présentation de la grande étude empirique que nous avons dédiée à ce scénario, très peu abordé dans la littérature SFP.

Chapitre 4 : Comparaison et évaluation empiriques des Systèmes Immunitaires Artificiels dans la Prédiction de défauts logiciels Inter-projets

Ce chapitre représente notre contribution principale dans la littérature SFP à travers une évaluation, qui se veut complète, des systèmes immunitaires artificiels (AIS), permettant de prédire les défauts logiciels avant qu'ils ne surviennent, dans un environnement inter-projets (entre des versions du même système) [HAO 20]. Nous allons essayer d'incorporer, dans ce chapitre, assez d'information pour la compréhension de ce domaine de recherche sans avoir à lire les chapitres précédents. Bien entendu, des redirections seront faites car nous voulons éviter de le surcharger par des répétitions inutiles. Ce chapitre n'est pas juste centré sur les AIS, mais beaucoup plus sur l'aspect inter-projets (PV) de l'étude, même si les algorithmes AIS jouent un rôle important dans cette investigation. Ils seront accompagnés de sept méthodes d'apprentissage souvent employées dans la littérature SFP afin d'avoir une vue générale du scénario PV.

4.1 Introduction

Les méthodes d'apprentissage automatique (ML) sont de plus en plus intégrées dans de nombreux domaines de recherche et d'ingénierie. Elles ne sont plus seulement exclusives à l'informatique. En effet, le génie logiciel s'avère être un domaine riche où plusieurs des tâches de développement (comme la phase de test logiciel) et de maintenance pourraient être formulées comme des problèmes d’apprentissage [ZHA 03-ABD 15-JUN 19].

Pour créer rapidement des logiciels complexes à la fois de qualité et fiables, nous appliquons plusieurs méthodes de contrôle au cycle de vie du développement (SDLC), ces techniques sont appelées méthodes d'assurance de qualité logicielle (Software Quality Assurance "SQA") (voir la section 2.2.1 chapitre 2). À ce propos, afin de garantir que le produit réponde parfaitement aux exigences formulées par le client, la méthode SQA la plus utilisée reste les tests logiciels (voir la section 2.2.2 du chapitre 2) [LES 08-ALT 17]. Cependant, la phase de test du cycle de vie est à la fois longue et fastidieuse, ce qui pourra conduire inlassablement à dépasser le budget et le temps alloués au développement du système [HAL 12-MAL 16-RAT 16].

Pour assister la tâche de test logiciel, la prédiction de défauts logiciels (SFP) rationalise l'effort et le budget dont le processus a besoin, en prédisant quelles sont les parties du nouveau programme (classes/composants/procédures) sujettes aux défauts ou non (fault-prone

et not fault prone), aidant ainsi les testeurs à se concentrer uniquement sur un nombre limité de module logiciels (modules sujets aux pannes) lorsqu'ils entament la phase de test [CAT 07 A-MEN 16].

Pour résumer, la prédiction de défauts systèmes consiste à construire des modèles de prédiction en utilisant, de préférence, des méthodes d'apprentissage automatique sur des données tirées de l'historique composés de métriques logicielles et des information relatives à des défauts d'un ancien programme, afin de classifier les instances du nouveau logiciel comme étant sujettes ou non à des bugs (voir la section 2.3.1 du chapitre 2) [KHO 03-JIA 08-HAL 12-RAD 13-KUM 18-AME 19]. Un processus SFP effectif pourrait résulter en un système avec une meilleure maintenabilités et refactoring, réduire le coût et ressource alloués, et développer ainsi un logiciel de qualité dans les temps impartis [CAT 11]. En outre, appliquer les modèles SFP à un stade avancé du SDLC comme les phases de spécification ou de conception logicielles, conduirait à améliorer son architecture et, éventuellement, atténuer le nombre de défauts dans le produit final [GLA 00-JIA 08].

Les études SFP peuvent être séparées en 2 grandes familles [MAL 16-HER 17-RAT 17]. La première Within-projects (au sein du même projet) elle-même catégorisée en deux autres scénarios qui sont la prédiction de défauts logiciels intra-projets (CV) et inter-projets (PV), où les travaux en CV représentent le sujet le plus dominant dans la littérature SFP. Pour édifier et valider un modèle de prédiction intra-projets, les données d'apprentissage et de test sont extraites du logiciel à analyser lui-même. Par contre, la validation des modèles PV s'appuie sur des instances tirées de deux versions différentes d'un même projet. Enfin, partant du postulat que les informations relatives aux défauts sont rarement conservées ou qu'il s'agit de la première version d'un nouveau projet [CAT 09-ZIM 09-HE 12-HER 18], la dernière famille de scénario SFP, s'intéresse à la construction de modèles de prédiction à partir de données calculés d'un projet (ou une de ses versions) complètement différent du logiciels à analyser. Ce dernier schéma d'étude s'appelle, la prédiction de défauts Cross-projets (CPDP). Pour plus de détails sur les scénarios de prédiction de défauts logiciels aller à la section 2.3.3 du chapitre 2.

Comme mentionné ci-dessus, pour construire un modèle de prédiction SFP, nous avons besoin de méthodes d'apprentissage automatique, afin de classifier automatiquement les modules d'un système en cours de développement comme étant sujets à des erreurs ou non. Certains algorithmes ont été largement évalués dans ce domaine tels que les réseaux de neurones (MLP), les arbres de décision (J48), Naïve Bayes (NB) et Random forest (RF), ainsi que des techniques de classification statistiques telles que la méthode de régression logistique (LR)[KHO 03-LES 08-CAT 09-HAL 12-MAL 15-RAT 17]. Par contre, la majorité des auteurs sont réticents à investiguer des algorithmes moins connus tels que les systèmes Immunitaires artificiels (AIS).

Historiquement parlant, l'humain a toujours été attiré par la nature et les interactions biologiques, s'en inspirant pour construire des outils afin de l'aider à survivre et s'adapter à son environnement. Même la science informatique à son lot d'approches développées à partir de la nature ou la biologie, ces techniques sont souvent étiquetées comme étant des méthodes

bio-inspirées. L'exemple le plus connu est celui des réseaux de neurones qui essaient de simuler le fonctionnement du cerveau humain à acquérir et sauvegarder les connaissances. Les algorithmes immunologiques, quant à eux, sont influencés par le système fascinant responsable de la protection de l'être vivant contre toutes sortes de maladies, virus, bactéries et des substances représentant un danger pour le bon fonctionnement de l'hôte [FOR 94-CAS 02-DAS 09]. Les méthodes AIS possèdent certaines caractéristiques souhaitables pour le développement d'algorithmes d'apprentissage automatique, telles que la mémoire, l'adaptabilité rapide, la décentralisation ainsi que le parallélisme [BRO 05].

Avant nos travaux [HAO 17-HAO 17 A-HAO 17 B], il n'existait que cinq travaux de recherche évaluant les systèmes AIS, principalement conduits par Catal et Diri [CAT 07-CAT 07 A-CAT 09 B], puis Abaei and Selamat [ABE 14] et plus récemment Kaur and Kaur [KAU 16]. Tous, sans exception ont effectué leurs investigations dans un scénario intra-projets, alors que 4 de ces travaux utilisent le même ensemble de données fournis par la NASA [SYA 5], seul le dernier [KAU 16] a employé des instances extraites de trois sous projets Eclipse composés de métriques de type processus appelés "micro-interaction metrics (MIM)" collectées par Lee et al [LEE 11]. Le résumé de ces travaux est exposé dans l'introduction du chapitre 3.

Un des objectifs principaux de ce chapitre est de retranscrire notre contribution principale [HAO 20], qui a pour but de comparer et évaluer les performances des systèmes AIS d'apprentissage automatique supervisé (voir chapitre 1) à construire des modèles de prédiction SFP inter-projets à travers trois benchmarks différents. Ainsi nous avons adopté 3 algorithmes AIS et leurs multiples versions dérivées élevant le nombre à 8 méthodes immunologiques (voir la section 1.5 du chapitre 1 pour avoir une idée sur le fonctionnement de ces algorithmes). Le premier benchmark consiste à comparer les performance des techniques AIS avec sept algorithmes populaires dans le domaine SFP, parmi eux le perceptron multicouche (MLP), les arbre de décision (J48), la Forêt d'arbres décisionnels (RF) et même une méthode de classification statistique qui n'est autre que la régression logistique (LR).

Selon Rathore and Kumar [RAT 17] très peu de travaux prennent en considération la prédiction de défauts inter-projets et davantage de recherches sont nécessaires pour évaluer l'utilité de ce genre de modèles. Comme les papiers d'intra-projets font légion dans le domaine SFP, par conséquent ils représentent une très bonne référence pour comparer et comprendre l'applicabilité des modèles PV.

Enfin, la troisième expérimentation ou plutôt benchmark, va permettre d'évaluer les modèles de prédiction de pannes logicielles PV, en comparant leurs performances à un certain critère proposé par He et al [HE 12]. Un modèle de prédiction est jugé efficace, si après validation, les valeurs des mesures de performances rappel (Recall) et Précision (Precision) atteignent 70% et 50% respectivement. Ainsi, pour construire et évaluer empiriquement les modèles étudiés, nous avons utilisés 41 bases de données open source se trouvant dans le référentiel de données PROMISE [MEN 16]. Ces instances sont le fruit des efforts de Jureczko et Madeyski [JUR 10] et Jureczko et Spinellis [JUR 10 A], elles sont caractérisées par 19 métriques logicielles orientéee objet, plus le nombre de lignes de codes (LOC).

Dans les études expérimentales appliquant les méthodes d'apprentissage artificiell (ML), il est difficile d'établir qu'elle algorithme à un avantage sur l'autre d'un point de vu performance, spécialement si plusieurs bases de données sont impliquées [LES 08-DEM 06]. De plus, pour avoir une expérimentation empiriquement fiable, les tests statistiques sont fortement préconisés [ARC 11]. Malhotra [MAL 16] a pointé du doigt que la signification statistique des résultats est rarement examinée dans ce domaine de recherche sans parler des modèles inter-projets. Pour cette raison, nous avons appliqué les tests statistiques lors de nos expérimentations, afin vérifier l'efficacité des modèles de prédiction PV construits.

À notre connaissance, il n'y aucune étude similaire utilisant cet ensemble de bases de données pour évaluer les algorithmes AIS dans la littérature de prédiction de défauts logiciels inter-projets. De plus, aucun des travaux cités dans le chapitre 3, employant les méthodes immunologiques, n'a cherché à comparer et évaluer méthodiquement les performances des technique AIS par des tests statistiques. Ainsi, dans notre démarche, nous avons appliqué le test statistique de Friedman [FRI 37], pour déterminé s'il y a une différence statistique entre les performances des 15 algorithmes ML que nous avons sélectionnés. Dans le cas où le test est concluant, nous effectuons une analyse Post-hoc en utilisant le test Nemenyi [NEM 63] dans le but de comparer par paire les performances des méthodes d'apprentissage évaluées. Les résultats de nos expérimentations sont formulés en 3 mesures de performances qui sont le rappel (Recall "PD"), Précision (Precision PR) et la f-mesure (F1-score) obtenus de la matrice de confusion après la validation des modèles (voir la section 2.5).

Les contributions de cette étude [HAO 20] peuvent être résumées ci-dessous :

- Améliorer la compréhension des systèmes Immunitaires artificiels dans la construction de modèles de prédiction de défaillances logicielles en général et plus précisément inter-projets, en utilisant une large sélection de bases de données (41) surtout en comparaison avec des études antérieures (5 était le maximum).
- Tenter de rafraichir ce sous ensemble d'études SFP inter-projets.
- Contribuer à l'état de l'art en menant une grande étude comparative composée de 15 algorithmes d'apprentissage automatique.
- Analyser les résultats par des tests statistiques pour valider empiriquement nos conclusions.

4.2 Genèse

Cette étude est le produit de 2 ans de travail dont le but initial était d'évaluer les systèmes immunitaires artificiels dans les 3 champs de la prédiction de défauts logiciels, en conduisant 3 expérimentations centrée, chacune, sur un des scénarios SFP possibles. Nous souhaitions évaluer de manière exhaustive les systèmes AIS et leurs performances vis-à-vis de la construction des modèles de prédiction logiciels. Pour cela, nous avons utilisé 41 bases de données issues de PROMISE, ce qui a conduit à la construction et la validation de 41 modèles de prédiction intra-projets pour chaque algorithme dans notre sélection, 30 modèles pour PV, ainsi que 1604 quand on se place dans un environnement Cross-projets. C'est l'expérimentation qui s'intéresse au problème CPDP qui nous a donné le plus de travail. Le

nombre élevé de modèles cross-systèmes est due au fait que nous avons appliqué une validation par paire (Pairwise) (voir la section 2.3.3 du chapitre 2).

Pour faciliter notre travail et gagner du temps, nous avons implémenté un outil pour automatiser la construction et les tests de ce nombre considérable de modèles SFP cross-projet. La durée d'exploitation pour chaque algorithme est comprise 17 minutes à 22 heures pour la méthode d'apprentissage gourmande SVM. Ce logiciel est une version simpliste du système que nous avons proposé dans la section 3.3 du chapitre 3, ce même programme est utilisé dans l'étude publiée [HAO 20] qui se concentre sur les problèmes SFP inter-projets et qui qui va être exposé en détails dans la suite de ce chapitre.

4.3 Motivation et Questions de recherche

D'après l'introduction du chapitre 3, nous pouvons soulever certains problèmes en ce qui concerne les travaux évaluant les systèmes immunitaires artificiels dans la prédiction de défauts logiciels :

- Toutes les études précédentes ne s'intéressent qu'au scénario SFP intra-projets.
- Malgré un nombre non négligeable de travaux dans ce domaine, rares sont ceux qui ont entrepris d'évaluer les AIS
- Toutes les études investiguant les méthodes immunologique n'emploient qu'un nombre réduit de bases de données (le max est de 5) pour généraliser leurs trouvailles.
- La fiabilité d'une expérimentation empirique ne peut être confirmée que par des tests statistiques. Cependant, les 5 travaux examinant les techniques AIS se gardent d'utiliser ce genre de méthodes, en ne basant leurs conclusions que sur la moyenne des mesures de performance établies, ce qui est insuffisant pour valider empiriquement leurs résultats [DEM 06-LES 08-ARC 11-MAL 16].

Pour répondre aux problèmes abordés au-dessus, nous avons accompli une large comparaison et évaluation empirique des systèmes immunitaires artificiels dans SFP. Notre étude présente les caractéristiques suivantes :

- Nous avons suivi une méthode de validation (test des modèles de prédiction) par paire (Pairwise) pour simuler le scénario inter-projet de prédiction des erreurs.
- Nous avons utilisé 41 bases de données issues du référentiel PROMISE représentant 11 projets différents.
- Pour valider empiriquement notre comparaison et évaluation, nous avons appliqué le test statistique de Friedman suivi par l'analyse Post-hoc par la méthode de Nemenyi, afin de comparer les performances des algorithmes sélectionnés, et pas seulement les systèmes AIS mais aussi 7 algorithmes ML référencés dans le domaine.
- Enfin, nous avons comparé les résultats obtenus dans le schéma inter-projets avec ceux extraits des modèles CV en employant le test de Wilcoxon sum-rank (test de la somme des rangs de Wilcoxon).

Nous avons formulé 4 questions de recherche (RQ) pour conduire nos expérimentations :

RQ1 : Lequel des algorithmes immunitaires artificiels est le plus adapté pour mener des études de prédiction de défauts logiciels inter-versions ?

La raison pour **RQ1** est d'avoir un petit aperçu sur quelle méthode immuno-inspirée est appropriée pour la construction des modèles de prédiction inter-projets, en utilisant une approche de validation par paire (Pairwise). Ramler et al [RAM 14] ont démontré que si un modèle est construit en employant les données de la version N du logiciel et testé par l'itération qui la suit directement N+1, les performances seront meilleures. Entre-autres, plus on s'éloigne de la version ciblée par l'analyse plus les résultats baissent. C'est pour cette raison que nous avons appliqué cette façon de valider nos modèles PV. Puis, les résultats obtenus, exprimés en 3 mesures de performance (Rappel, Précision et F1), devraient être évalués par le test de Friedman, ainsi qu'analysés par la méthode Post-Hoc de Nemenyi.

RQ2 : Dans quelle mesure les systèmes immunitaires artificiels fonctionnent-ils bien par rapport à des algorithmes référencés pour la prédiction de défauts logiciels inter-projets ?

Cette question est justifiée par le fait que nous n'aurons pas assez de perspective sur les performances des systèmes AIS si on ne les compare pas à d'autres méthodes d'apprentissage référencées dans le domaine. Cette expérimentation suit les mêmes étapes expliquées pour répondre à RQ1, mais cette fois nous ajouterons 7 algorithmes ML bien connus de la littérature.

RQ3 : Dans quel scénario de prédiction de défauts logiciels within-project (au sein du même projet) les systèmes immunitaires artificiels sont-ils les plus performants ?

Nous savons tous que les études les plus populaires dans SFP sont les travaux en Intra-projets (CV), ce qui représentera une très bonne expérimentation pour évaluer les modèles PV construits par les systèmes immunologiques. Pour simuler le schéma de prédiction des défauts CV, nous avons appliqué la validation croisée "N-fold cross validation" (voir la section 2.3.3 du chapitre 2) où N=10 sur les 15 algorithmes d'apprentissage de notre sélection. La comparaison entre les modèles PV et CV est faite par le test statistique Wilcoxon sum-rank (test de la somme des rangs de Wilcoxon).

RQ4 : Les modèles intra-projets construits remplissent-ils le critère de performance postulés par He et al. [HE 12], c'est-à-dire avoir au moins 70 % de Rappel (Recall), et 50 % de Précision ?

Nous voulions par cette expérimentation comparer et évaluer nos modèles de prédiction inter-projets à un critère de performance généralement accepté dans la littérature SFP, pour déclarer que le système a bien fonctionné. Ce critère est lui-même, une version plus simple du seuil proposé par Zimmermann et al [ZIM 09]. Où, les modèles construits doivent atteindre des performances mesurées par Recall et Précision dépassant les 75%. Selon He et al [HE 12] seuls 18 sur 34 modèles de prédiction cross-projets ont remplis leur critère de performance.

A notre connaissance, aucune recherche publiée n'a évalué les systèmes immunitaires automatiques dans un environnement de prédiction de défauts logiciels inter-projets. Ce qui nous motive à produire une étude qui se veut complète, afin d'évaluer et comparer les AIS dans ce scénario.

4.4 L'approche proposée pour notre étude empirique

Pour répondre aux questions de recherches formulées dans la section 4.3 et valider empiriquement les résultats obtenus, nous devons suivre une certaine approche. D'abord, nous devons sélectionner les données appropriées (section 4.4.1), puis, choisir les bonnes méthodes d'apprentissage et comment seront validés les modèles construits (section 4.4.2 et section 4.4.4). Enfin, nous devons évaluer et comparer ces modèles en utilisant la mesure de performance adéquate (section 4.4.3) et le test statistique adapté à ce genre d'études (section 4.4.5). La *Figure 4.*1 résume schématiquement l'approche que nous avons suivie.

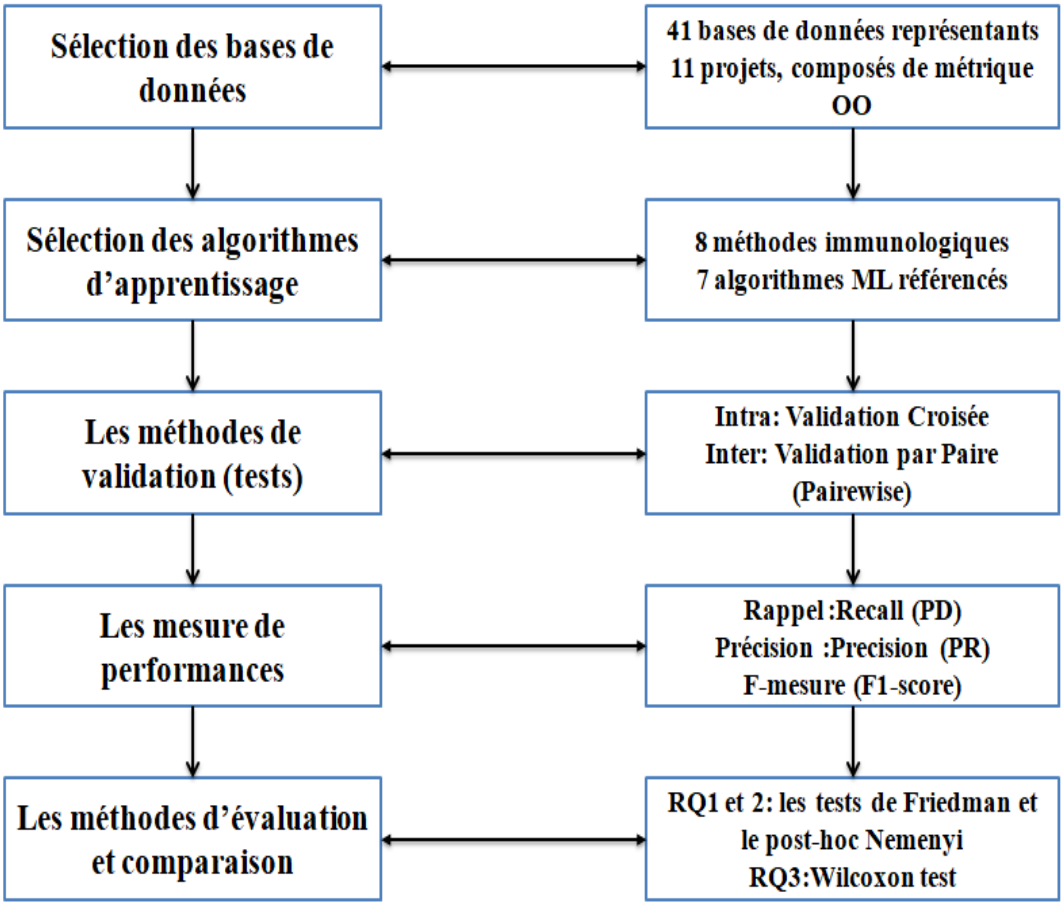


Figure 4. 1- Résumé de notre approche empirique

Le reste de cette section, va aborder les bases de données que nous avons employées ainsi que les métriques logicielles qui les composent. Par la suite, une petite description des algorithmes d'apprentissage choisis pour cette expérimentation afin d'évaluer les systèmes AIS. Enfin, les méthodes de validation, mesures de performance et les statistiques nécessaires à accomplir notre évaluation empirique seront détaillés.

4.4.1 Bases de données et métriques logicielles

Comme dans le chapitre 3, nous allons utiliser des bases de données libres d'accès qui se trouvent dans le référentiel de données PROMISE[13] [MEN 16]. Mais cette fois au lieu de cinq bases, 41 d'entres-elles (*Table 4.*1) seront sélectionnées afin de conduire nos expérimentations. Elles sont issues des travaux de Jureczko et Madeyski [JUR 10] ainsi que Jureczko et Spinellis [JUR 10 A] qui donnent plus d'information sur la manière avec laquelle les données sont rassemblées. Les logiciels open source d'où sont extraites ces données, appartiennent à The Apache Software Foundation, ils sont tous écrits en langage Java.

***Table 4.* 1-Bases d'apprentissages utilisés dans notre étude**

Version	#Classes	% Défauts		Version	#Classes	%Défauts
Ant-1.3	125	16		**lucene-2.2**	247	58
Ant-1.4	178	22		**lucene-2.4**	340	60
Ant-1.5	293	11		**poi-1.5**	237	59
Ant-1.6	351	26		**poi-2.0**	314	12
Ant-1.7	745	22		**poi-2.5**	385	64
camel-1.0	339	3		**poi-3.0**	442	64
camel-1.2	608	36		**synapse-1.0**	157	10
camel-1.4	872	17		**synapse-1.1**	222	27
camel-1.6	965	19		**synapse-1.2**	256	34
ivy-1.1	111	57		**velocity-1.4**	196	75
ivy-1.4	241	7		**velocity-1.5**	214	66
ivy-2.0	352	11		**velocity-1.6**	220	35
jedit-3.2	272	33		**xalan-2.4**	723	15
jedit-4.0	306	25		**xalan-2.5**	803	48
jedit-4.1	312	25		**xalan-2.6**	885	46
jedit-4.2	367	13		**xalan-2.7**	909	99
jedit-4.3	492	2		**xerces-1.2**	440	16
log4j-1.0	135	25		**xerces-1.3**	453	15
log4j-1.1	109	34		**xerces-1.4**	588	74
log4j-1.2	205	92		**xerces-init**	162	48
lucene-2.0	195	47				

La raison pour laquelle nous avons choisi d'utiliser ces données est qu'aucun papier étudiant les AIS dans le domaine SFP n'a employé ces instances, sauf nos travaux antérieurs

[13] http://openscience.us/repo

[HAO 17-HAO 17 A-HAO 17 B]. Mais plus encore, cet ensemble de données contient de multiples versions d'un même projet ce qui serait parfait pour une étude SFP inter-projet. Le même prétraitement des données suivi dans l'expérimentation du chapitre 3 section 3.2 a été appliqué dans ce travail (Voir la section 3.2.1). Enfin, le fait que ces données soient publiquement disponibles assurera que notre étude soit répétable et nos résultats seront fiables et vérifiables.

Chacune des instances de ces bases de données est composée de 20 métriques orientées objets (OO), plus précisément 19 mesures OO et le nombre de lignes de codes (LOC), plus la variable dépendante Bug qui contient le nombre de défauts recensé par classe. Ces métriques logicielles sont listées dans la table 3.2 du chapitre 3 section 3.2.1. Pour plus d'information sur ces mesures et leurs rôles aller au chapitre 2, section 2.4.

4.4.2 Algorithmes d'apprentissages Automatiques

Rappelons qu'un des buts de notre recherche est d'évaluer la capacité des systèmes AIS à construire des modèles de prédiction de défauts logiciels inter-projets. Donc nous avons sélectionné 8 méthodes immunologiques (AIRS 1, AIRS2 AIRS2.parallel, CLONALG, CSCA, Immunos-1, Immunos-2, Immunos-99) et comparé leurs performances avec 7 algorithmes référencé dans le domaine SFP. La liste de ces techniques est présentée dans la *Table 4*.2. Les détails des algorithmes AIS sont, quant à eux, explorés dans la section 1.5 du chapitre 1.

Enfin, pour assurer la reproductibilité de notre expérimentation, les paramètres d'utilisateur qui aident à ajuster les algorithmes sont laissés par défauts à l'exception du Kernel de la méthode SVM (le Kernel par défauts dans WEKA c'est RBF). Toutes les méthodes que nous avons citées sont implémentées dans l'outil WEKA (section 2.9 chapitre 2) puis importées à notre logiciel (Section 4.2) qui nous a aidés à mettre en oeuvre nos expérimentations.

4.4.3 Mesures de performance

Dans la section 2.5 du chapitre 2, nous avons parlé longuement des métriques avec lesquelles les modèles de prédiction de défauts logiciels sont évalués. Tout part de la matrice de confusion ci-dessous :

	true	false
true	TP	FN
false	FP	TN

Matrice de confusion

Où :

- ***TP :*** le nombre de vrais positifs qui représente le nombre d'exemples sujets aux défauts qui sont classés comme tels.
- ***FN :*** le nombre de faux négatifs qui représente le nombre d'exemples étiquetés fault prone mais classés faux par le modèle.

- ***FP:*** le nombre de faux positifs (faux rejet), exemples non sujets aux défauts mais classés vrais .
- ***TN :*** le nombre de vrais négatifs, qui représente le nombre d'exemples sans défauts classés comme tels par le modèle.

À partir des nombres ci-dessus, nous pouvons ainsi calculer nos métriques de performances :

- **Le taux des exemples bien classés (Puissance / Accuracy) =** $\frac{TP+TN}{TP+FP+TN+FN}$
- **Rappel (Recall « PD ») =** $\frac{TP}{TP+FN}$ Mesure la probabilité qu'un module sujet aux pannes soit correctement classé.
- **Précision (Precision « PR ») =** $\frac{TP}{TP+FP}$ elle montre le nombre de modules prédits correctement comme sujets aux défauts parmi tous les modules prédits défectueux.
- **F-mesure (mesure F/F1-score)=** $2 * \frac{(PD*PR)}{(PD+PR)}$

Le choix de la mesure à utiliser reste un débat continuel dans la communauté SFP, tout dépend de préférence de l'auteur. Dans notre cas, nous avons choisi 3 métriques de performances qui sont le Rappel (PR), Précision (PD) et la f-mesure (f1). Cette sélection est due au fait que ces 3 mesures portent des propriétés assez intéressantes du point de vue d'un praticien. En effet, plus le modèle validé a une valeur PR élevée plus le système a réussi à prédire les défauts existants dans le projet cible. Tandis qu'un score haut pour la métrique Précision signifirait que la phase de test sera réduite. La f-mesure représente la moyenne harmonique entre le Recall et Précision, ainsi F1 exprime un compromis entre PD et PR, plus exactement, entre le nombre de modules sujets aux pannes et le temps à passer lors des tests [CAT 12-KAU 15].

4.4.4 Méthodes de validation

Pour évaluer les performances des modèles de prédiction inter-projets exprimées en Rappel, PD et F1, nous allons suivre une méthode de validation classique par paire (Pairewise), où les données d'apprentissage et de test sont tirées de deux versions différentes d'un même projet. Par exemple, si le modèle est construit par la version Camel 1.4 il sera testé (validé) par les instances de la version Camel 1.5. Avec nos 41 bases de données sélectionnées qui représentent 11 projets open source, nous aurons à construire et valider 30 modèles pour les 15 algorithmes étudiés.

De plus, aucune des premières versions de chaque projet ne va servir de base de test et vice versa, aucune des plus récentes ne va être pour l'apprentissage.

Table 4. 2- Liste des techniques d'apprentissages référencées

Algorithme	Description
Arbre de décision (J48)	Cette technique construit un arbre où les attributs (métriques) les plus efficaces pour classer les modules défaillants sont choisis à chaque nœud selon un certain critère de division. C4.5 utilise l'entropie et le gain d'information.
K-plus proches voisins (IBK)	IBK cherche l'instance d'apprentissage la plus proche (KNN) de celle testée à l'aide d'une simple distance Euclidienne. Le résultat de la recherche représentait la classification de l'instance inconnue [WIT 16]. KNN est également utilisé pour la phase de classification dans de nombreux algorithmes AIS (AIRS 1-2 et parallèle AIRS2, CSCA).
Régression logistique (LR)	C'est algorithme le plus utilisé et aussi l'un des meilleurs pour les études en SFP. LR est une méthode de classification statistique qui utilise la fonction de régression logistique pour estimer la probabilité de classification d'une instance donné.
Le Perceptron Multicouche (MLP)	La méthode d'apprentissage automatique bio-inspirée de référence par excellence. les détails de cet algorithme sont fournis dans la section 1.3.2 du chapitre 1.
Classifieur Bayésien Naïf (NB)	NB est un classificateur probabiliste basé sur le théorème de Bayes avec l'hypothèse que chaque paire d'attributs (métriques logicielles) est indépendante l'une de l'autre. Cette hypothèse peut être considérée comme simpliste. Cependant, le classificateur Naive Bayes a été utilisé avec succès dans de nombreux travaux SFP.
Forêt d'arbres décisionnels (RF)	L'objectif est d'édifier une forêt d'arbres de décision, en sélectionnant au hasard les attributs d'entrée pour diviser le nœud et seul le meilleur est adopté. Chaque arbre de la forêt est cultivé sans élagage. Enfin, dans la phase de classification, tous les arbres participent à l'étiquetage de l'instance inconnue par un vote majoritaire où Random Forest (RF) choisit la classe qui a le plus de votes.
Machine à vecteurs de support (SVM)	Cette technique utilise le concept d'hyperplan pour séparer l'ensemble de données en instances sujettes et non sujettes aux défauts. Dans le cas d'une région non linéaire, nous aurons besoin d'une transformation des données dans un espace de dimension supérieure à l'aide de fonctions noyau (Kernel). Après plusieurs tests et les résultats montrés par les travaux de [KAU 15], nous avons choisi d'utiliser le Kernel polynomial pour avoir les meilleures performances possibles.

Pour établir la deuxième benchmark de notre étude, il fallait construire des modèles intra-projets (CV) pour savoir dans quel scénario SFP les méthodes étudiées sont les plus adaptées. Ainsi, les 41 modèles de prédiction nécessaires pour chaque algorithme sont validés par la méthode habituelle propre à ce scénario dans la littérature. Ainsi, l'apprentissage et le test suivent une approche de validation croisée (N-fold cross-validation) où les données de chaque

base sont séparées aléatoirement en N parties. Généralement, N est égal à 10, ainsi 9 des N parties vont servir de données d'apprentissages et la dernière comme étant des instances de test. Ce processus va être répété jusqu'à ce que tous les groupes aient joué le rôle de données de test et apprentissage. Toutes les informations à propos des méthodes de validation pour chaque scénario SFP sont expliquées dans la section 2.3.3 du chapitre 2.

4.4.5 Les tests statistiques

Généralement, les travaux évaluent les méthodes AIS en utilisant que la valeur moyenne des mesures de performances. Par contre, la fiabilité des résultats de l'étude empirique serait entachée si l'analyse n'a pas appliqué les tests statistiques pour valider les conclusions. Demsar [DEM 06] préconise l'utilisation des tests non paramétriques pour comparer les prestations des algorithmes d'apprentissage dans le cas où l'étude utilise plusieurs bases de données. Cette vue a été chaudement supportée par Lessmann et al [LES 08], ainsi que Malhotra [MAL 16] dans leur contribution dans le domaine SFP.

Les tests non-paramétriques s'affranchissent de fait que les données suivent une distribution normale ou que les variances exhibées par les performances des modèles de prédiction sur les bases de données soient similaires. Par contre, ces tests sont moins puissants que leurs cousins paramétriques tels que le test T.

Ainsi, pour remplacer le test d'analyse de la variance (ANOVA "analysis of variance"), pour comparer les performances de nos multiples méthodes ML étudiées, nous avons appliqué le test de Friedman [FRI 37]. Ce test octroie une valeur moyenne des rangs à toutes les méthodes par rapport à leurs performances, dans notre cas, selon les valeurs des mesures Rappel, Précision et F-1. Plus le rang est bas, plus la méthode est performante. L'hypothèse (H0) de ce test suppose qu'il n'y a aucune différence statistiquement significative entre les résultats des algorithmes. Dans le cas où H0 est rejetée, alors il y a quelque part une distinction entre les performances des modèles de prédiction. Cet algorithme a un degré de liberté de N=Nombre d'algorithmes-1 basé sur la distribution χ^2 (chi-square). Cette statistique est calculée selon l'équation suivante :

$$\chi^2 = \frac{12}{NK(K+1)} \sum_{i=1}^{K} R_i^2 - 3N(K+1) \quad (4.1)$$

Où, K représente le nombre de bases de données, tandis que R_i est le rang alloué au $i^{ème}$ algorithme par le test et N est le nombre d'algorithmes-1.

Dans le cas où le test de Friedman produit un résultat significatif, nous rejetons H0 dans le cas où la valeur p (p-value) est inférieure à 5%. Ainsi, nous appliquerons l'analyse Post-hoc par le test Nemenyi comme préconisé par Demsar [DEM 06], Lessmann et al [LES 08] et Malhotra [MAL 16].

Le test de Nemenyi compare, par paires, les performances des algorithmes étudiés en se basant sur les mesures de performances employées. Pour cela, il calcule une valeur de distance critique (CD) entre les moyennes des rangs fournis par le test de Friedman aux méthodes à comparer. Si la différence entre les deux rangs est supérieure ou égale à CD, alors

les performances entre les deux algorithmes sont statistiquement différentes. Pour calculer CD il faut utiliser:

$$CD = Q_\alpha \sqrt{\frac{N(N+1)}{6K}} \text{ (4.2)}$$

Où, Qα est l'écart studentisé divisé par √2.

Enfin, nous avons utilisé le test Wilcoxon sum-rank (test de la somme des rangs de Wilcoxon /Mann Whitney U Test) [MAN 47] pour répondre à notre troisième question de recherche (RQ 3) afin de voir dans quel scénario SFP les méthodes AIS sont les plus adaptées. Le test évalue la signification en calculant la différence entre les résultats des techniques de validation intra et inter-projets résultants de des modèles construits, puis les classe en fonction de leurs valeurs absolues. Le processus commence par regrouper en conséquence de la différence entre PV et CV, pour un algorithme, en un seul échantillon combiné, en gardant une trace d'où provient chaque observation. Après, il sont triés du plus bas 1, au plus élevé N, où N est le nombre d'échantillons.

La statistique U du test de Mann et Whitney est la petite valeur de U1 et U2 calculés comme suit :

$$U1 = N1N2 + \frac{N1(N1+1)}{2} - R1 \text{ (4.3)}$$

$$U2 = N1N2 + \frac{N2(N2+1)}{2} - R2 \text{ (4.3)}$$

Où, R1 et N1 sont la somme des rangs, ainsi que le nombre d'échantillons pour un groupe (par exemple Inter-projets). R2 et N2 sont pour l'autre groupe.

Dans la suite de ce chapitre, nous allons présenter et discuter les résultats que nous avons obtenus après les expérimentations que nous avons exposées dans cette section.

4.5 Résultats et discussion des expérimentations

Cette section présente un résumé des résultats que nous avons obtenus. Dans ce travail, nous avons réalisé nos expérimentations sur un ordinateurs portable HP Pavilion-g6 équipé d'un processeur Intel i7-3632QM avec 2,20 GHz et 6,00 Go de RAM, exploités sous Windows 10. De plus, les API de l'outil *Weka* et le *langage R* sont utilisés pour mener l'étude, le dernier est employé pour effectuer les tests statistiques. Pour automatiser la construction et la validation des modèles de prédiction, nous avons développé un petit logiciel permettant de réaliser les expériences souhaitées (section 4.2). Les sous-sections suivantes contiennent les réponses aux questions de recherche formulées dans la section 4.3.

4.5.1 Lequel des algorithmes immunitaires artificiels est le plus adapté pour mener des études de prédiction de défauts logiciels inter-versions ?

Pour évaluer l'efficacité des systèmes AIS dans l'environnement de prédiction des défauts logiciels Inter-projets (PV), nous avons utilisé trois mesures de performance, à savoir la

précision (PR), le rappel (PD) et la mesure F (F1-score). Un résumé des résultats est présenté dans la *Table 4.*3, que nous avons clairement illustré dans les diagrammes en boîte (Box-plot) des *Figures* 4.2 à 4.4. Ces illustrations montrent les performances des 30 modèles construits et testés en appliquant la technique de validation pour les travaux SFP inter-projets expliquée dans la section 4.4.4 pour chaque algorithme de notre sélection (section 4.4.2). Les prestations pour chaque mesure de performance sont exposées en termes Max, Min et Moy. Où Max représente la meilleure performance atteinte par un algorithme ML parmi ses 30 modèles validés, alors que Min c'est pour le pire résultat prélevé, tandis que Moy est la moyenne des performances.

Tous ce qui est en gras dénote les meilleures performances enregistrées sur entre tous les algorithmes d'apprentissage listé dans la section 4.4.2. Par contre, si la valeur moyenne de la mesure est en italique c'est pour ressortir la méthode AIS la plus adaptée pour la construction des modèles PV. Les résultats détaillés sont présentés dans l'Annexe A.

***Table 4.* 3- Résumé des résultats des modèles de prédiction inter-projets**

	Précision			**Rappel**			**F-mesure**		
Algorithme	Moy	MAX	MIN	MOY	MAX	MIN	MOY	MAX	MIN
AIRS1	0.443	1.000	0.000	0.403	0.930	0.000	0.347	0.772	0.000
AIRS2	0.489	1.000	0.087	0.395	0.965	0.032	0.367	0.789	0.061
AIRS2. parallèle	*0.502*	1.000	0.075	0.363	0.944	0.025	0.350	0.773	0.045
CLONALG	0.378	1.000	0.000	0.370	1.000	0.000	0.305	0.778	0.000
CSCA	0.465	1.000	0.000	0.385	0.875	0.000	0.342	0.765	0.000
Immunos 1	0.385	0.988	0.030	***0.780***	1.000	0.174	***0.453***	0.899	0.057
Immunos 2	0.237	1.000	0.000	0.292	1.000	0.000	0.201	0.798	0.000
Immunos 99	0.370	0.988	0.000	0.734	1.000	0.000	0.426	0.912	0.000
J48	0.492	1.000	0.000	0.430	0.937	0.000	0.386	0.773	0.000
NB	0.553	1.000	0.082	0.399	0.887	0.097	0.405	0.752	0.134
MLP	0.520	1.000	0.094	0.444	0.930	0.028	0.392	0.772	0.054
RF	0.521	1.000	0.000	0.427	0.937	0.000	0.387	0.778	0.000
SVM	0.423	1.000	0.025	0.511	0.911	0.069	0.369	0.748	0.049
LR	**0.560**	1.000	0.095	0.406	0.923	0.025	0.381	0.768	0.044
IBK	0.495	1.000	0.071	0.444	0.901	0.025	0.399	0.759	0.037

D'après la *Table 4.*3 et la *Figure 4.*2, il semblerait que la méthode AIRS2 parallèle a la meilleure performance parmi les techniques AIS sur les 30 modèles de prédiction avec une moyenne de PR=0.502. Suivie directement par les algorithmes CSCA et AIRS 2 avec un score de précision égale à 0.465 et 0.489 respectivement. Concernant la mesure de performance PR, les 3 versions de la méthode Immunos-81 ont les plus faibles rendements.

Une autre observation montre que presque toutes les méthodes immunologiques sont parvenues à un score PR parfait de 100 % (Précision= 1) surtout quand les données d'apprentissage sont extraites du logiciels Xalan 2.6 et que le test est effectué sur les instances de la version suivante (Xalan 2.7). Ce qui voudrait dire que le processus de test serait léger, où le testeur ne ciblera que les modules logiciels classés comme sujets à des défauts (fault-

prone). Même si ces résultats sont très encourageants, certains modèles construits par les algorithmes AIS affichent une valeur minimale de Précision =0 (exemple où la version 1.2 du système Camel joue le rôle des données de tests), montrant l'incapacité du modèle à faire la différence entre les classes sujettes aux défauts et celles qui ne contiennent aucune erreurs. Ainsi, le testeur subira un processus très lourd.

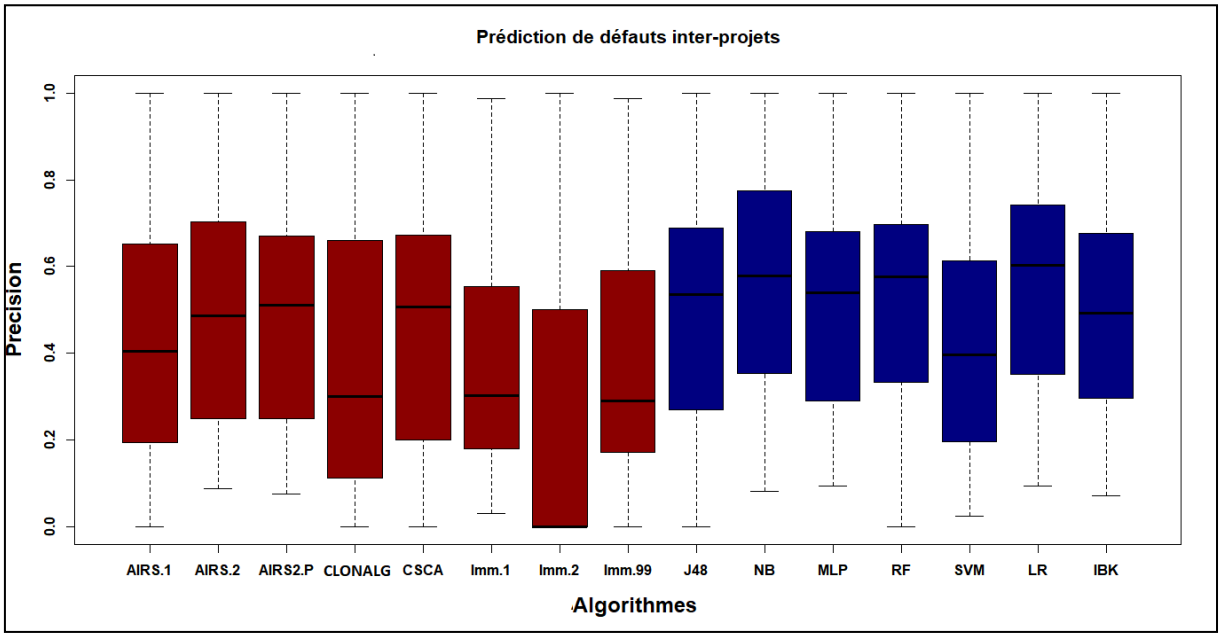


Figure 4. 2- diagrammes en boîte de la mesure Précision (PR)

Dans le cas où la comparaison est faite par rapport à la mesure de Rappel (PD "Recall"), la *Figure 4.*3 montre clairement que la méthode Immunos-1 a le meilleur rendement possible avec un ratio presque égal à 80%. Suivie directement par la troisième version du même algorithme avec PD=0.734. Par contre, Immunos-2 reste très loin de ces standards, et enregistre les pires performances représentées par les valeurs des mesures PR et Recall avoisinant une moyenne de 0.292 sur les 30 méthodes validées.

Les autres techniques AIS ont des résultats moyens entre 0.363 et 0.403 où la méthode AIRS 1 est la plus efficace, alors que AIRS2.parallèle est la moins adaptée. Notons que les performances de cet algorithmes parallèle pourraient être meilleures ou encore pire que ce qui a était rapporté, car sur 10 exécutions et validations du même modèle PV par cet algorithme, nous avons 10 différents résultats. C'est pour cette raison, que nous avons pris la valeur médiane pour chaque modèle construit par AIRS 2.parallèle après 10 exécutions.

Enfin, tout comme pour la mesure PR, certains algorithmes AIS ont pu prédire tous les défauts d'un logiciel analysé tel qu'Ant-1.5, Poi-3.0 et Lucene- 2.4. Comme le montre la valeur maximale 1 de la métrique Rappel surtout par les techniques Immunos-1 et 99. Ceci signifie que ces méthodes immunologiques seraient adéquates pour analyser ou prédire les défauts de logiciels critiques [KAU 15].

En ce qui concerne la troisième mesure de performance, les rendements des méthodes AIS varient entre 0.201 et 0.453, où la plus haute performance est au nom de Immunos-1 suivie par Immunos-99 avec une valeur moyenne de la f-mesure (F1) =0.425. Par conséquent, Si la personne de l'équipe de développement, responsable de la prédiction des défauts système, veut avoir des modèles représentant un bon compromis entre une grande détection des défauts et un test logiciel plus léger, alors il n'aura qu'à choisir une de ces deux versions de l'algorithme Immunos-81 pour construire ces modèles inter-projets.

La meilleure performance possible par un système immuno-inspiré exprimée avec la métrique F1 a été obtenue par la méthode Immunos-99 avec un score de 0.912, montrant qu'on est arrivé à prédire presque tous les défauts possibles, ainsi qu'à réduire grandement notre effort de test pour le projet Log4j-1.2. Par contre, ce n'est pas toujours aussi évident, car presque toutes les techniques AIS ont une valeur Min de la mesure F égale à 0, suggérant que ne n'avons pas pu prédire les erreurs possibles que pourrait contenir un logiciel, ce qui implique un passage par un processus de test assez lourd, spécialement si le modèle construit par des données de la version 1.4 du programme Ivy.

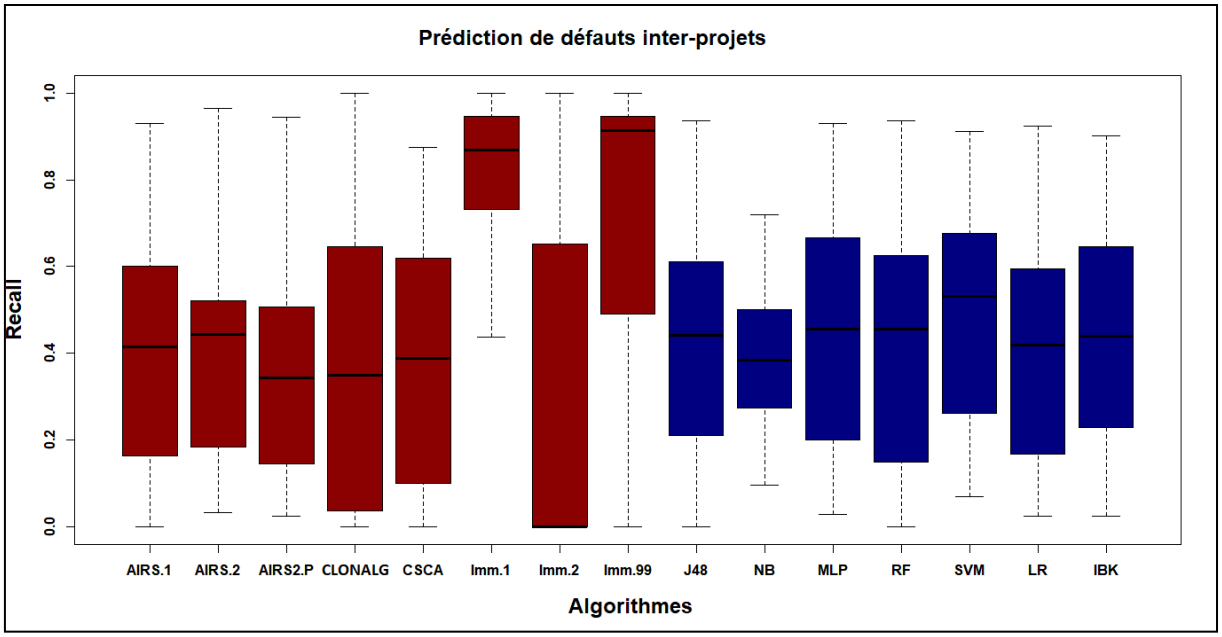


Figure 4. 3- Diagrammes en boîte de la mesure Rappel (PD)

Selon un autre point de vue, c'est que le choix de l'algorithme d'apprentissage qui joue un rôle primordial pour l'édification d'un bon modèle de prédiction inter-projets. Par exemple si la version 1.1 du logiciel Log4j est la cible de l'analyse SFP, le modèle validé par Immunos-2 a un score F1 de 0, alors que, si AIRS1 est utilisé, ce score serait 0.646, renforçant l'idée que la sélection de l'algorithme adapté à ce genre d'étude est tout aussi importante que les bases de données employées.

Selon Demsar [DEM 06], le fait de se contenter des valeurs moyennes pour évaluer les performances des classifieurs en la présence de multiple base de données n'est pas empiriquement fiable, l'addition de tests statistiques est nécessaire afin de vérifier les résultats

de la comparaison. Dans notre démarche, nous avons utilisés le test de Friedman pour estimer la différence de rendement entre les systèmes AIS. Ce test fournis les rangs combinés des modèles de prédiction construits et validés par toutes les méthodes d'apprentissages automatiques sélectionnés (8 techniques AIS et 7 autres algorithmes ML) à partir de base de données extraites de 11 projets et leurs multiples versions choisis pour nos expérimentations se trouvant dans PROMISE. Cela nous permettra de déterminer de manière fiable que la différence entre les systèmes AIS est statistiquement significative. Ainsi, établir une fois pour toutes, laquelle des méthodes d'apprentissage automatique immunologique est la plus adaptée pour la recherche PV.

Le test Friedman n'est pas seulement appliqué sur les méthodes AIS, mais plutôt sur tout l'ensemble des algorithmes que nous avons sélectionnés portant le nombre à 15 techniques. Par contre, dans cette section nous n'allons comparer que les méthodes immuno- inspirées dans la construction des modèles de prédiction inter-projets. L'hypothèse nulle (H0) du test stipule que tous les algorithmes étudiés ont les mêmes performances. Nous nerejetterons H0 que si le test est statistiquement significatif, où nous avons fixé la valeur de rejet (P-value) communément acceptée dans le domaine de 5% (la valeur P< 0.05).

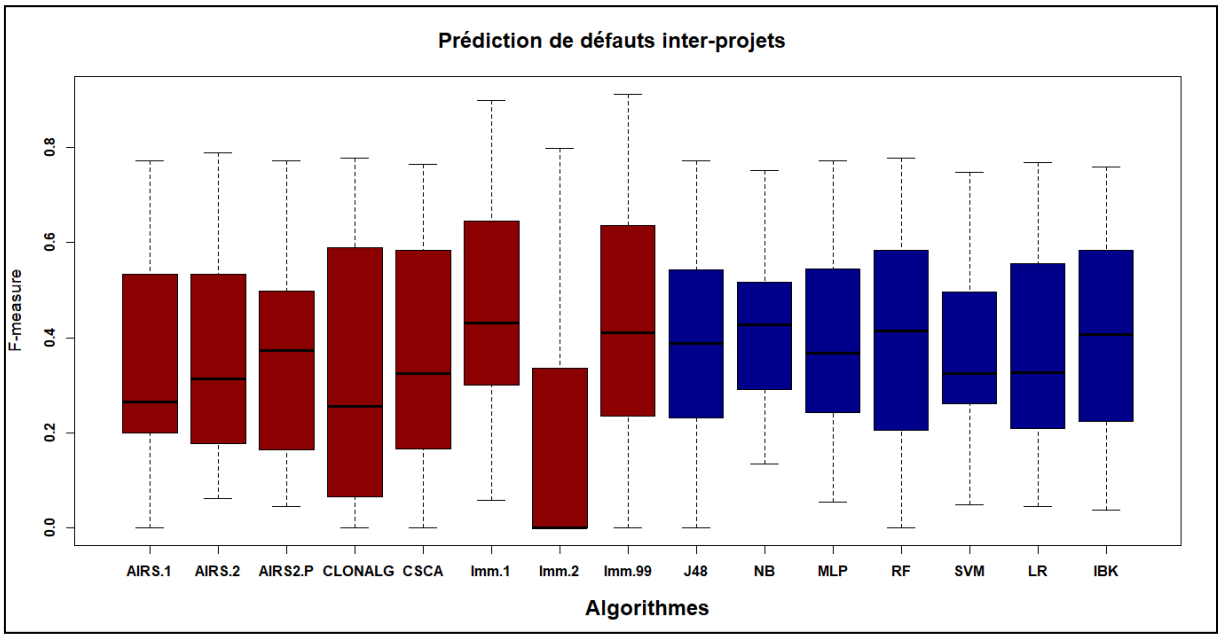


Figure 4. **4- Diagrammes en boîte de la F-mesure (F1-score)**

Comme nous sommes en train de comparer 15 méthodes ML, alors le degré de liberté est égal à 14 (N-1). Les valeurs statistiques du test sont 161.37 pour la précision, 101.2 pour le Recall et enfin 56.91 pour F1. Ainsi, les P-value pour chaque mesure de performance sont égales à $2.2*10^{-16}$ pour PR, $2.792*10^{-15}$ pour PD tandis que pour F1 c'est $3.3959*10^{-17}$.

Après avoir vu les résultats du test non paramétrique de Friedman, nous pouvons rejeter H0 et accepter l'hypothèse alternative (H1) qu'il y a au moins une différence significative entre paire d'algorithmes comparés. Pour chercher quelles sont les paires responsables de ce rejet, nous avons appliqué l'analyse post-hoc de Nemenyi, afin de déterminer avec précision quelle méthode AIS est meilleure que l'autre. La moyenne des rangs résultants du test de

Friedman sont présentés dans la *Table 4.*4. Tandis que la *Table 4.*5 affiche le bilan de l'analyse Post-hoc. Où 0 voudrait dire qu'il n'y pas de différence significative entre la paire analysée, alors P signifie le contraire en terme de mesure de performance Précision (PR), R c'est pour le Rappel (PD) et F est liée à la f-mesure(F1). Sur les 105 paires évaluées, seules 37 sont statistiquement significatives pour PR, 21 pour le Recall et seulement 8 pour F1.

Table 4. **4- la moyenne des rangs du test Friedman**

	AIRS1	AIRS2	AIRS2.P	CLONALG	CSCA	Immunos 1	Immunos 2	Immunos 99	J48	NB	MLP	RF	SVM	LR	IBK
PR	9,0	7,4	6,9	8,6	7,0	11,7	12,1	12,2	5,8	4,0	5,8	4,9	10,3	**4,3**	6,6
PD	8,7	8,4	9,9	9,7	9,3	**2,7**	10,4	**3,8**	7,9	8,6	7,1	7,3	6,3	8,3	6,5
F1	9,5	8,4	9,5	9,4	9,1	**6,2**	12,0	7,2	6,9	6,7	6,6	6,7	8,5	7,1	**6,2**

D'après les tables 4.4 et 4.5, AIRS 2.parallèle est le meilleur algorithme parmi les systèmes immunitaires artificiels, suivi directement par CSCA et la version 2 du même algorithme, avec un rang moyen de 7 sur 15 pour la mesure de performance Précision. Toutes les itérations de la méthode Immunos-81 sont les moins performantes avec un classement de 12 en moyenne pour la métrique PR. L'analyse Post-hoc suggère qu'il n'y a pas de différence significative parmi les systèmes AIS sauf dans le cas de la paire Immunos-1 et 2. Ce qui rend difficile de choisir quelle méthode AIS est plus adaptée pour l'étude des modèles de prédiction inter-projets en se basant sur la mesure de performance PR.

Table 4. **5- Le résultat de l'analyse Post-hoc**

	AIRS.2	AIRS2.P	CLONALG	CSCA	Imm.1	Imm.2	Imm.99	J48	NB	MLP	RF	SVM	LR	IBK
AIRS.1	0	0	0	0	**R**	0	**R**	0	**P**	0	**P**	0	**P**	0
AIRS.2	-	0	0	0	**P R**	**P**	**P R**	0	0	0	0	0	0	0
AIRS2.P	-	-	0	0	**P R**	**P**	**P R**	0	0	0	0	0	0	0
CLONALG	-	-	-	0	**R**	0	**R**	0	P	0	0	0	**P**	0
CSCA	-	-	-	-	**P R**	**P**	**P R**	0	0	0	0	0	0	0
Imm.1	-	-	-	-	-	**R F**	0	**P R**	**P R**	**P R**	**P R**	0	**P R**	**P**
Imm.2	-	-	-	-	-	-	**R F**	**P F**	**P F**	**P F**	**P F**	**R**	**P F**	**P F**
Imm.99	-	-	-	-	-	-	-	**P R**	**P R**	P	**P**	0	**P R**	**P**
J48	-	-	-	-	-	-	-	-	0	0	0	**p**	0	0
NB	-	-	-	-	-	-	-	-	-	0	0	**p**	0	0
MLP	-	-	-	-	-	-	-	-	-	-	0	**p**	0	0
RF	-	-	-	-	-	-	-	-	-	-	-	**p**	0	0
SVM	-	-	-	-	-	-	-	-	-	-	-	-	**p**	0
LR	-	-	-	-	-	-	-	-	-	-	-	-	-	0

Par contre, quand l'évaluation utilise la métrique Rappel, le verdict est sans appel, Immunos-1 et Immunos-99 sont de loin les meilleurs algorithmes AIS. Où le rang en moyenne de la 1ère version est presque égal à 2 tandis que pour la version hybride cela avoisine les 4 mais il n'y a pas de différence significative entre les deux. Les algorithmes restants n'ont qu'un classement varient de 8.7 à 10.7 où la méthode Immunos-2 est bonne dernière.

Enfin, l'analyse des performances en utilisant la métrique f-mesure traduit un rang de 6.2 pour la méthode Immunos-1 suivi par les 7.2 d'Immunos-99 les rendant encore une fois, les meilleurs algorithmes immunologiques en termes de F1. Par contre l'analyse post-hoc n'est pas concluante et seules les paires Immunos-1 et 2 ainsi que Immunos-99 et la version 2 sont statistiquement significatifs. Ceci confirme que l'algorithme Immunos-2 n'est pas vraiment adapté pour l'étude de la prédiction de défauts logiciels, bien que les travaux antérieurs [CAT 09 B- ABA 14] affirment que la technique marche bien en la présence de métriques logicielles orientée objets.

Réponse 1 (RQ1) : la méthode AIS la plus adaptée pour l'étude des modèles de prédiction défauts logiciels inter-projets, dépend grandement des besoins du manageur du projet. Si le but est de réduire l'effort de test, alors il faut choisir un modèle avec une valeur de Précision élevée [JIA 08-A38-BRO 11]. La méthode AIRS2.parallèle a le meilleur rendement dans ce cas-ci. Par contre, nous recommandons l'utilisation des autres versions du même algorithme ou à la rigueur la méthode CSCA, car AIRS2.parallèle produit des résultats non consistants tout le long de nos expérimentations.

Quand notre besoin est de détecter ou prédire le plus de modules logiciels sujets à des défauts (c'est important surtout les systèmes critiques) [JIA 08-A38-BRO 11], nous devons choisir un modèle qui a atteint le plus haut score pour la mesure de performance Rappel. D'après notre évaluation, il est clair que la technique Immunos-1 est la plus efficace, mais le point noir à cette conclusion, c'est que la méthode n'a pas de paramètres utilisateur qui nous aideraient à affiner l'algorithme pour améliorer ses performance (voir le chapitre 1 section 1.5.3). Ainsi, nous recommandons l'utilisation de la version hybride du même système.

En outre, l'analyse des performances des méthodes AIS par rapport à la mesure F1 n'est pas statistiquement pertinente, même si Immunos-1 à la valeur moyenne la plus élevée, mais nous ne pouvons pas nous prononcer avec certitude sur les résultats de cette comparaison avec cette métrique de performance. Enfin, la seconde version de l'algorithme Immunos-2 parait inadaptée pour la prédiction de défauts logiciels PV, se classant dernière par rapport à toutes les mesures de performances sélectionnées.

4.5.2 Dans quelle mesure les systèmes immunitaires artificiels fonctionnent-ils bien par rapport à des algorithmes référencés pour la prédiction de défauts logiciels inter-projets ?

Pour mieux évaluer la fiabilité des systèmes immunitaires artificiels à prédire les défauts logiciels dans environnement inter-projets, nous avons comparé les algorithmes AIS à 7 méthodes d'apprentissage automatiques référencées dans le domaine SFP. Ces techniques largement utilisés dans la littérature sont listés dans la *Table 4.*2. Le résumé des résultats obtenus est présenté dans la *Table 4.*3 et les *Figures* 4.2 à 4.4.

Quand la comparaison s'appuie sur la métrique de performance PR, LR à la valeur moyenne la plus élevée sur les 30 modèles construit atteignant les 0.560, suivi par NB avec Précision= 0.553. Néanmoins, le rang affiché par la *Table 4.*4 de NB est meilleur que celui de la régression logistique avec 4 et 4.3 respectivement. Par contre, l'analyse Post-hoc montre qu'il n'y a pas de différence significative entre les deux techniques représentées dans la *Table 4.*5. Plus encore, la *Table 4.*4 montre une moyenne de rang très bas de 10.3 par un algorithme aussi populaire que les SVM, dépassé par 5 méthodes AIS qui sont les 3 versions de AIRS, ainsi que les deux versions de l'algorithme inspiré de la sélection clonale avec un classement variant de 6.9 à 9 avec le test de Friedman.

D'un autre côté, l'analyse conduite par le test de Nemenyi entre les SVM et les 5 méthodes AIS n'est pas concluante en utilisant ce type de base de données. Cela voudrait dire que, dans certain cas, nous pourrons replacer l'algorithme SVM pas ces 5 méthodes immunologiques surtout pour gagner en temps d'exécution. En effet, pour construire et valider les 30 modèles de prédiction PV par la méthode SVM, il nous a fallu 7 heures par accomplir cette tâche, mais ça n'a duré que 11 minutes pour AIRS 2. Enfin, même si ces résultats sont encouragaents pour l'utilisation des systèmes AIS dans la prédiction de défauts logiciels, il n'est pas encore temps d'abandonner les autres algorithmes, car les résultats des 3 versions de la méthode Immunos-81 ont un faible taux de Précision, se classant entre 11.7 et 12.2, insinuant que ces techniques n'arrivent pas souvent à faire la différence entre les modules sujets aux défauts et ceux qui ne comportent pas de pannes sous-jacentes.

Étonnamment, quand le but du chef de projet est de prédire le plus grand nombre de défauts possible pour un logiciel donné, il faudrait un modèle avec des valeurs de Rappel très élevées ; le résultat obtenu dans notre cas reflète presque l'opposé de notre conclusion basée sur la mesure PR. C'est l'algorithme SVM qui a le meilleur taux de Rappel ainsi que le meilleur rang parmi les méthodes non AIS, avec PD= 0.511 et classement 6.3. Suivi directement par IBK avec un rang de 6.5 comme l'indique la *Table 4.*4. Par contre, si on regarde très bien la *Figure 4.*3 nous allons voir clairement que le meilleur ou au moins la méthode qui a le plus haut classement est Immunos-1. Encore plus étonnant, l'analyse Post-hoc affiché dans la *Table 4.*5 montre qu'Immunos-1 à des différences significatives avec presque toutes les techniques sélectionnées pour notre expérimentation sauf la version hybride du même algorithme, qui est une meilleure alternative qu'Immunos-1 (puisque ce dernier ne présente aucun paramètre d'utilisateur). Par contre, les autres méthodes AIS occupent le bas du tableau et Immunos-2 est le pire d'entre eux. Les autres méthodes non immuno- inspirées se classent entre 7.1 pour MLP et 8.6 pour le Naïve Bayes.

Concernant, la comparaison des algorithmes AIS avec les 7 méthodes d'apprentissage référencés en employant la f-mesure, il en ressort que Immunos-1 et IBK ont les meilleurs

rangs d'après le principe du test Friedman qui est égale à 6.2 pour tous les deux, ainsi qu'une moyenne de F1=0.453 pour Immunos-1 et 0.399 pour IBK (k-plus proche voisins). Même si que Immunos-99 à un taux plus élevé de f-mesure que IBK, sont classement est plus bas. Par contre, il représente une meilleure alternative qu'Immunos-1 à cause des problèmes que nous avons cités auparavant. Tous les autres algorithmes AIS partagent, encore une fois, la seconde partie de tableau (*Table 4.*4) où leur classement commence par 8.4 (AIRS2) et se termine par Immunos-2 et une moyenne de rang égale à 12 sur 15. L'analyse post-hoc n'offre pas plus de détails afin de situer les performances des systèmes immunitaires dans la prédiction de défauts PV en se basant sur F1-score.

Réponse 2(RQ2) : Quand on considère l'exploration des performances des algorithmes AIS par rapport à des méthodes ML de référence (*Table 4.*2) dans le scénario inter-projets de la prédiction des défaillances logicielles. Les systèmes immunitaires sont plus faibles que leurs homologues en termes de la métrique Précision. Néanmoins, certaines méthodes immunologiques ont un rendement compétitif avec une technique aussi populaire que SVM, représentant une meilleure alternative que cette dernière, surtout si en prend en considération les temps d'exécution.

Par contre, les AIS se rachètent fortement quand la comparaison est portée par la mesure Rappel. Où les deux meilleurs algorithmes, et de loin, sont des méthodes AIS, plus précisément, Immunos-1 et Immunos-99. Ce qui voudrait dire que ces deux versions de la technique Immunos-81 sont très adaptées pour prédire le plus de défauts possibles dans un schéma SFP inter-projets.

Enfin, l'analyse post-hoc de Nemenyi, montre que 14 des quinze algorithmes choisis, ne montrent aucune différence statistiquement significative entre eux. Ainsi, il sera difficile de mettre une méthode devant l'autre dans cas où la mesure de performance est F1.

4.5.3 Dans quel scénario de prédiction de défauts logiciels within-project (au sein du même projet) les systèmes immunitaires artificiels sont-ils les plus performants ?

Le but de cette section est de déterminer, si les performances des systèmes immunitaires à construire des modèles de prédiction inter-projets est comparable à celles du scénario intra-projets (CV). De ce fait, nous avons conduit une expérimentation pour évaluer les modèles intra-projets sur les 41 bases de données sélectionnées, suivant la méthode de validation expliquée dans la section 4.4.4. Les résultats de cette expérience sont résumés dans la *Table 4.*6. Enfin, un aperçu de cette comparaison est accessible dans la *Figure 4.*5 pour la Précision, la *Figure 4.*6 pour le Rappel et la *Figure 4.*7 pour la f-mesure.

Les résultats sont présentés sous forme de maximum (MAX), minimum (MIN) et la valeur moyenne (MOY) sur les 41 modèles construits pour chacune des mesures de performances.

Ajouter à cela, la valeur P-value du test Wilcoxon qui compare les modèles PV et CV de chaque algorithme. La valeur en gras représente la meilleure performance obtenue pour une métrique donnée sur les 15 méthodes d'apprentissages étudiées, par contre l'italique est réservé pour ressortir la technique AIS la plus adaptée.

Table 4. 6-Résumé des résultats pour la prédiction de défauts logiciels Intra-projets

	Précision				Rappel				F-mesure			
Algorithme	MOY	MAX	MIN	P-value	MOY	MAX	MIN	P-value	MOY	MAX	MIN	P-value
AIRS1	0.502	0.993	0.000	0.269	0.485	0.980	0.000	0.211	*0.491*	0.986	0.000	**0.018**
AIRS2	0.548	1.000	0.000	0.322	0.464	0.991	0.000	0.306	0.488	0.992	0.000	**0.043**
AIRS2.P	*0.557*	0.991	0.000	0.334	0.443	0.989	0.000	0.228	0.483	0.990	0.000	**0.035**
CLONALG	0.455	0.988	0.000	0.249	0.374	1.000	0.000	0.775	0.398	0.994	0.000	0.232
CSCA	0.538	0.988	0.000	0.300	0.430	1.000	0.000	0.489	0.459	0.994	0.000	0.076
Immunos 1	0.416	0.997	0.065	0.549	***0.820***	1.000	0.265	0.442	0.480	0.842	0.118	0.506
Immunos 2	0.328	1.000	0.000	0.400	0.355	1.000	0.000	0.636	0.312	0.994	0.000	0.376
Immunos99	0.414	0.992	0.043	0.376	0.742	1.000	0.211	0.496	0.471	0.829	0.077	0.388
J48	0.565	0.996	0.000	0.295	0.508	0.999	0.000	0.237	0.526	0.997	0.000	**0.015**
NB	0.584	0.996	0.143	0.576	0.440	0.918	0.188	0.343	0.471	0.915	0.172	0.115
MLP	0.575	0.989	0.125	0.481	0.521	1.000	0.077	0.233	0.542	0.994	0.095	**0.006**
RF	**0.631**	0.994	0.000	**0.044**	0.516	1.000	0.000	0.205	**0.554**	0.997	0.000	**0.004**
SVM	0.417	0.993	0.067	0.944	0.538	0.989	0.091	0.977	0.453	0.991	0.083	0.169
LR	0.595	0.994	0.000	0.572	0.480	0.991	0.000	0.297	0.517	0.992	0.000	**0.024**
IBK	0.548	0.994	0.000	0.422	0.533	0.944	0.000	0.161	0.540	0.968	0.000	**0.009**

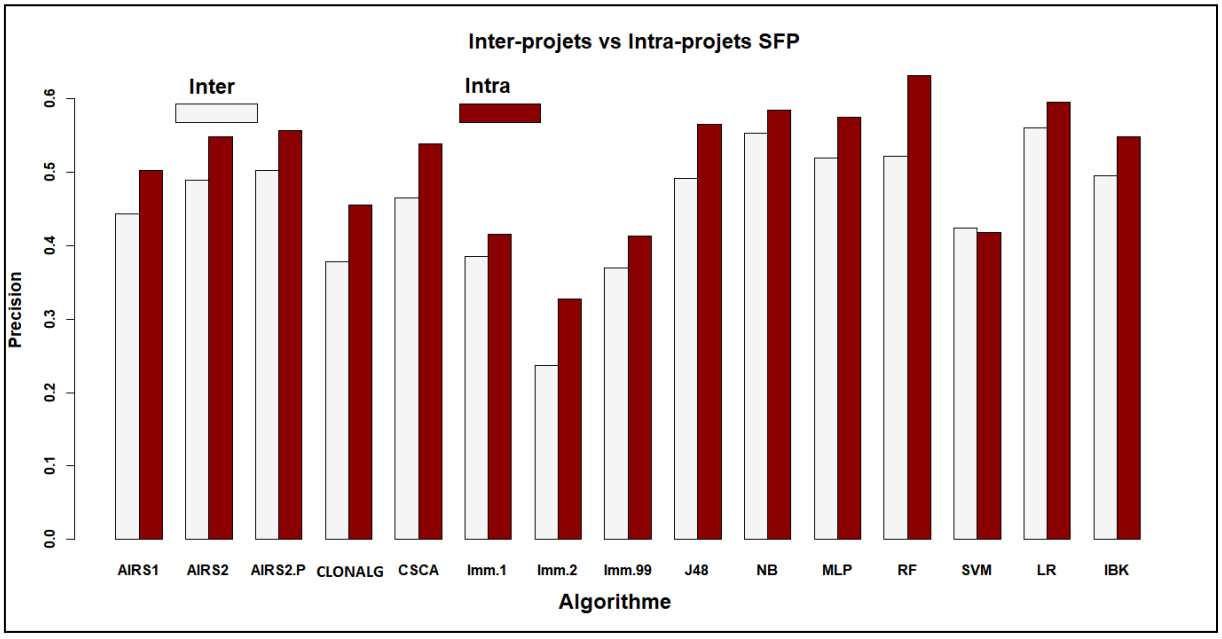


Figure 4. 5- La comparaison des modèles intra et inter-projets avec la mesure Précision

L’étude des rendements fournis par les modèles CV pour les 15 algorithmes ML reflète presque à l'identique à ceux établies par le scénario inter versions, en tout cas pour les deux premières mesures de performances (PR et PD). Où le meilleur algorithme selon la *Table 4.*6 est Immunos-81 avec une moyenne de 82% pour les 41 modèles validés par cet algorithme par rapport à la métrique Rappel. Immunos-99 viens directement après lui avec PD=0.742.

Les autres techniques viennent loin derrière, en commençant par la méthode SVM et son score de 53.8 % jusqu'à l'algorithme le moins adapté Immunos-2 avec 35.5%. La seule différence à noter, entre les scénarios intra et inter projets, correspond aux performances de RF en ce qui concerne la mesure Précision, où son rendement avoisine une moyenne de PR=0.631, alors que c'était LR le plus performant dans le schéma PV.

La méthode AIRS2 parallèle achève à l'identique l'observation trouvée dans l'étude des modèles PV, comme étant la meilleure technique à inspiration immunologique pour la réduction de l'effort du test lors du cycle de développement. Cependant, nous conseillons d’utiliser les autres versions du même algorithme à cause son instabilité due à la primitive "Merge" qui forme la cellule mémoire à partir des différents threads de la méthode (Voir la section 1.5.1 chapitre 1).

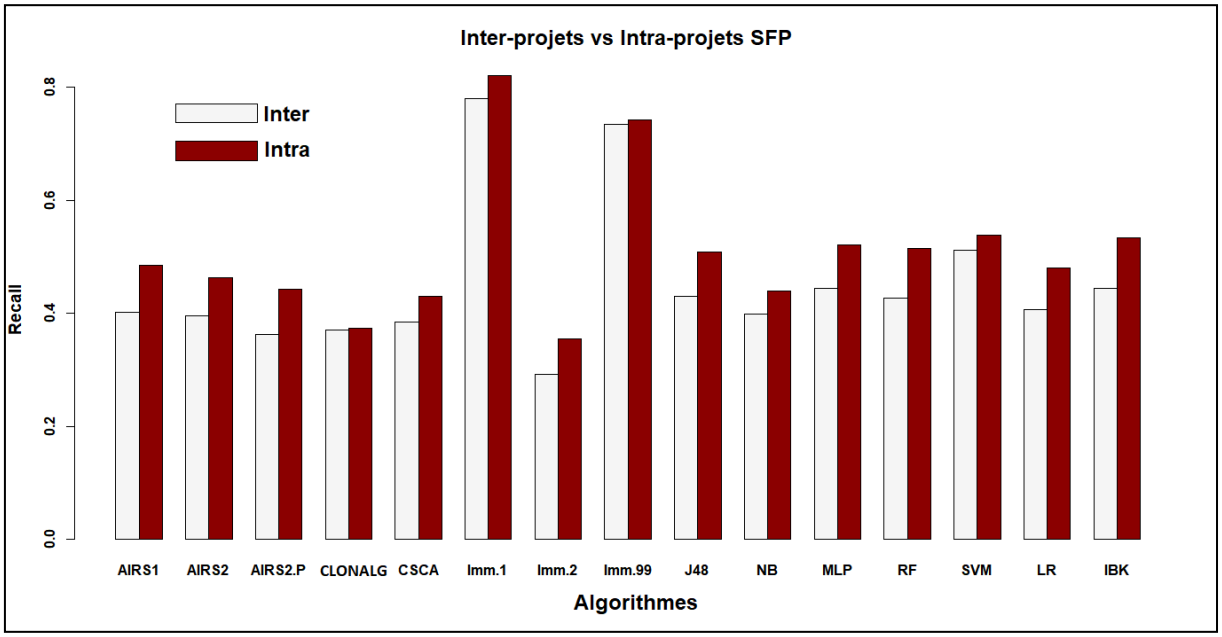


Figure 4. **6-La comparaison des modèles intra et inter-projets avec la mesure Rappel**

Concernant la mesure F1, l'algorithme RF est en tête avec une moyenne de 0.554 sur les 41 modèles intra-projets, alors que pour PV c'était Immunos-1 qui était quatrième parmi les méthodes AIS, derrière les 3 versions de la technique AIRS. Pour Immunos-2 c'est toujours le même constat, bon dernier, il semble, de plus en plus, non adapté à ce genre d'études surtout si nous ajoutons le fait que ce programme n'a pas de paramètre d'utilisateur pour aider à ajuster ces performance.

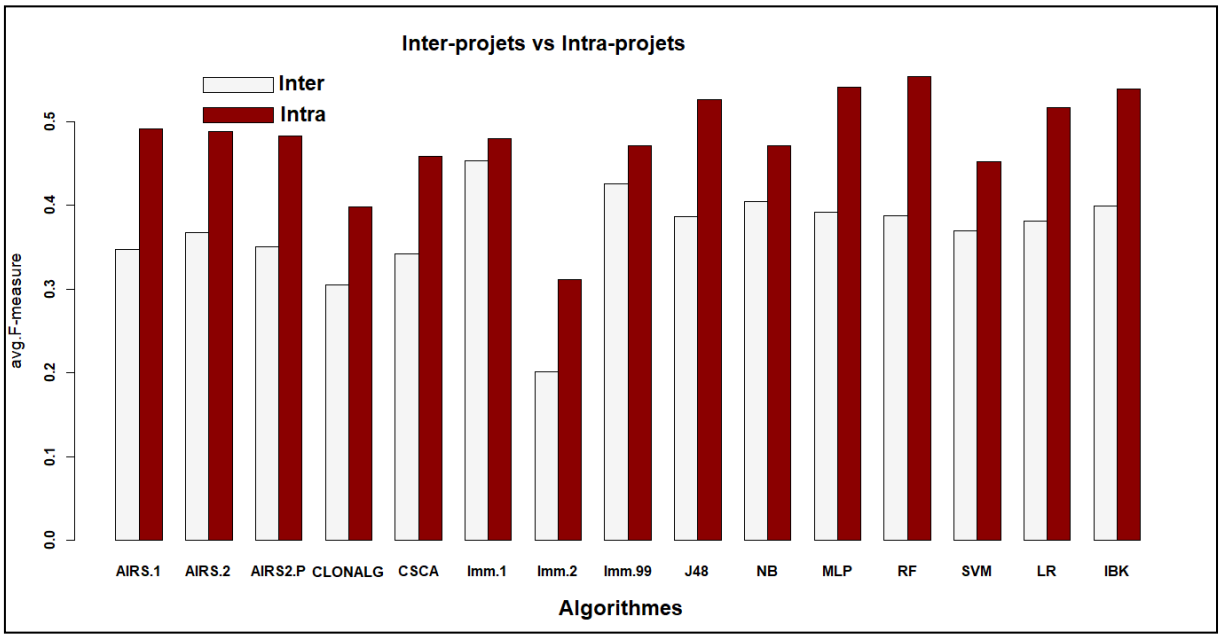


Figure 4. 7- La comparaison des modèles intra et inter-projets avec la f-mesure

À partir des *Figures* 4.5 à 4.7, nous pouvons voir clairement que les performances des modèles CV ont l'avantage sur ceux produits par le scénario inter-projets, sauf un cas particulier, où PV est meilleur que CV représenté par la mesure Précision avec la méthode SVM. Par contre, cette différence devient plus perceptible quand la f-mesure est utilisée. Néanmoins, ces analyses ne sont basées que sur les valeurs moyennes des métriques de performances, ce qui n'est pas très fiable surtout quand l'étude est portée par de multiple base de données.

Par conséquent, nous avons appliqué le test statistique de Wilcoxon pour comparer les modèles des deux scénarios SFP sur les 3 mesures de performance choisies. Ce test a pour hypothèse nulle (H0) que les performances de l'algorithme analysé sont les mêmes que ce soit dans un schéma inter-projets ou intra-projet. Les résultats de la méthode Wilcoxon sont affichés dans la *Table 4.*6 sur la colonne P-value. Pour rejeter H0, nous avons choisi le seuil habituel de 5%.

Excepté pour RF, nous ne pouvons pas rejeter l'hypothèse nulle suggérant que les modèles construit par les algorithmes sont les même peu importe le scénario étudié si notre analyse s'arrête qu'à la mesure Précision. Cependant, 8 des 15 méthodes évaluées ont des différences significatives par rapport à la métrique F1, trois d'entre elles sont des algorithmes AIS, plus précisément les méthodes AIRS1, AIRS2 et AIRS2.parallèle. Par contre, les deux techniques que nous avons établies comme meilleures pour la construction des modèles inter-projets ne sont pas statistiquement différentes pour les modèles CV. Ainsi, Immunos-1 et Immunos-99 restent stables peu importe le scénario SFP dans lequel on se place.

Réponse 3 (RQ3): l'investigation pour répondre à notre troisième question de recherche (RQ3), montre que les performances entre les deux scénarios within-projets (au sein du même projet) sont généralement comparables et peuvent être intervertis, même si les valeurs moyennes des mesures de performances sont un peu meilleures pour CV que pour PV.

Néanmoins, cette différence n'est pas statistiquement significative selon le test de Wilcoxon Mann et Whitney que nous avons appliqué. Ce qui contredit les conclusions de Rathore and Kumar [RAT 16], et qui est plutôt similaire à celles de Malhotra [MAL 16].

4.5.4 Les modèles intra-projets construits remplissent-ils le critère de performance postulé par He et al. [HE 12], c'est-à-dire avoir au moins 70 % de Rappel (Recall), et 50 % de Précision ?

Dans cette expérimentation, nous voulions comparer et évaluer les systèmes immunitaires artificiels à un seuil accepté dans les travaux SFP pour juger si les modèles construits ont une quelconque utilité pratique. Il y a dans la littérature deux critères de performances pour considérer que le système à bien fonctionné et atteint ses objectifs. Le premier défini en 2009 par Zimmermann et al [ZIM 09], où les deux mesures de performances (Rappel et Précision) doivent dépasser 75% de réussite, jugé très restrictif et haut par certains papiers dans le domaine [KUM 18-HER 17-HE 12-HER 18]. Dans l'année 2012, He et al [HE 12] ont suggéré une version moins contraignante de ce critère, où le Rappel des modèles validés doit au moins parvenir à 70% (0.7) et 50% pour la Précision. Le résultat de ce critère est présenté dans la *figure* 4.8.

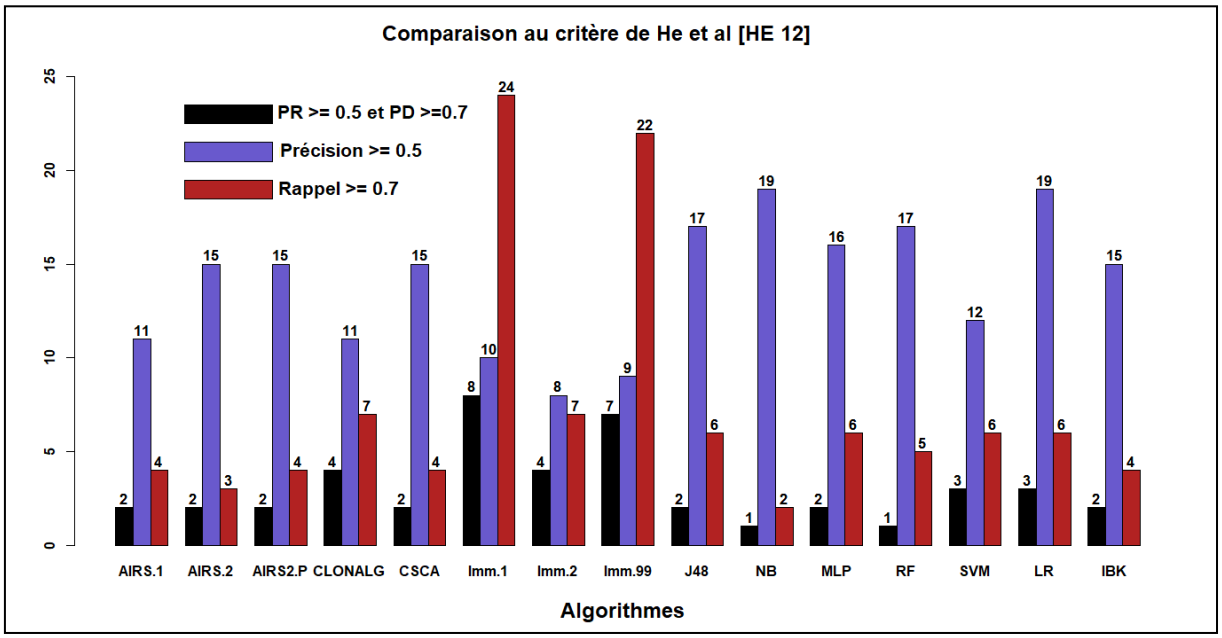


Figure 4. 8- **Évaluation des modèles de prédiction de défauts logiciels inter-projets selon le critère de performance proposé par He et al [HE 12]**

La figure 4.8 reflète que très peu de modèles ont rempli le critère souhaité par He et al [HE 12]. La meilleure performance à cet égard est attribuée à la méthode AIS Immunos-1 avec seulement 8 modèles qui ont une utilité pour les praticiens, suivi par Immunos-99 avec 7. Étonnamment, les deux algorithmes Immunos-2 et CLONALG, malgré leur faible rendement d'après les deux précédentes benchmarks, ont réussis à remplir le contrat surpassant même des

algorithmes référencés tels que MLP, SVM et RF alors que LR n'a que 3 modèles qui ont atteint le critère demandé.

Une autre observation que nous pouvons tirer de *la Figure 4.8*: C'est que Immunos-1 et la version hybride du même système ont atteint 24 et 22 fois respectivement le critère de la mesure Rappel, montrant, encore une fois, l'importance du choix de la méthode d'apprentissage appropriée pour faire une étude de prédiction de défauts logiciels. Plus le score de PD est haut, plus le système a prédit des défauts, ce qui est essentiel pour le développement des systèmes critiques. D'un autre côté, NB et la régression logistique sont arrivés à atteindre le critère de Précision de 0.5, 19 fois sur 30 modèles. Ce qui résulterait à un effort de test plus léger lors du cycle de vie du logiciel analysé. D'ailleurs, AIRS2.parallèle et CSCA ont aussi accomplis des modèles remplissant le seuil PR à 15 occasions.

Réponse 4 (RQ4) : L'évaluation et comparaison des systèmes d'apprentissage immunologiques, en considérant le critère de performance proposé par He et al [HE 12], démontre à quel point tous les algorithmes sélectionnés sont loin de parvenir aux performances désirées. Immunos-1 et Immunos-99 ont le meilleur rendement d'après ce seuil.

L'importante observation à l'égard de cette analyse est l'impact du choix de l'algorithme approprié pour l'édification des modèles de prédiction PV afin d'atteindre les objectifs fixés par le chef de projet. Si la prédiction du plus grand nombre de défauts est la priorité alors nous revendiquons l'utilisation de la méthode Immunos-99, car la première version du même algorithme souffre de l'absence des paramètres utilisateur pour réajuster le système et améliorer ses performances.

Par contre, si le but est d'alléger l'effort du test logiciel alors il faudrait choisir entre l'algorithme de classification statistique LR et la méthode NB, ainsi que le système AIRS2 pour construire les modèles de prédiction de défauts inter-projets.

4.6 Menace de validité

Dans cette section nous allons exposer certaines menaces qui pourraient nuire à la validité de nos conclusions.

La Validité des conclusions : concerne la relation entre le traitement effectué et le résultat obtenu. Dans cette étude nous avons utilisé des tests statistiques non paramétriques (Friedman,Nemenyi et Wilcoxon) car ils ne nécessitent pas certaines préconditions pour bien fonctionner. Telles que : les données analysées doivent suivre une distribution normal ou l'homogénéité de la variance. De plus, nous avons évalué ces résultats par rapport un seuil accepté par la communauté de la recherche SFP de 5%. Enfin, le rendement a aussi subi une analyse post-hoc renforçant encore plus notre confiance sur la validité de nos conclusions [LES 08-MAL 16-DEM 06-HE 15].

La validité interne : Comme notre étude n'utilise que des algorithmes présents dans l'outil open source de WEKA sans toucher aux paramètres d'utilisateur restés par défaut, alors il n'y pas de menace interne dans cette partie.

Les résultats pourraient être différents si nous avions employé d'autres métriques de performance. Dans nos expérimentations, nous avons utilisé 3 différentes mesures que nous trouvons très intéressantes du point de vue des praticiens [JIA 08-A38-BRO 11]. Un haut score de Rappel (PD) signifierait une très bonne prédiction des modules logiciels sujets aux défauts. Une très bonne valeur de Précision (PR) se traduirait par un effort de test logiciel réduit. Enfin, la f-mesure représente un compromis entre les deux métriques précédentes.

Validité externe : elle s'intéresse à la généralisation des résultats obtenus [HE 15]. La menace principale vient des données sélectionnées. Rappelons que nous utilisons 41 bases de données, représentant 11 projets open source, extraites du référentiel de données publiques PROMISE. Par conséquent, les résultats fournis par notre expérimentation sont acceptables vu la nature publique des instances utilisées, la liberté d'accès à ces données aiderait à refaire et vérifier facilement nos conclusions. Par contre, les futures recherches dans le domaine devraient chercher à analyse les modèles SFP sur des projets développés dans d'autres langages de programmation pour la généralisation des rendements affichés par l'étude effectuée.

Conclusion et Perspectives

Notre travail se place dans le cadre général de l'ingénierie logicielle empirique (empirical software engineering) dans l'optique d'utiliser des méthodes de fouille de données bio-inspirées pour des problèmes liés à l'ingénierie logicielle. Notre choix se porte sur le domaine de la prédiction de défauts logiciels (SFP) en utilisant des méthodes bio-inspirées. Vu que le domaine SFP est assez actif, et que les systèmes immunitaires artificiels (AIS) sont très peu étudiés dans ce cadre, il nous a paru intéressant d'étudier l'apport de ce genre de méthodes qui semblent bien adaptées à ce type de problèmes.

Après des lectures approfondies dans la littérature du domaine, nous avons commencé par conduire trois études expérimentales [HAO 17-HAO 17 A-HAO 17 B] où nous avons d'abord évalué 8 algorithmes AIS dans un scénario SFP intra-projets pour construire des modèles de prédictions de défauts logiciels d'abord sur 5 puis sur 10 bases de données extraites du référentiel PROMISE [MEN 16]. Nous avons aussi proposé un outil d'aide à la prédiction de défauts dans un scénario Cross-projects pour aider les praticiens à introduire la prédiction de défauts logiciels dans leur cycle de développement. Cet outil nous a permis d'améliorer le taux de classification grâce à l'incorporation d'un système de vote pour l'étiquetage des instances du système analysé.

Le but principal de notre recherche est d'évaluer empiriquement les méthodes d'apprentissage automatique d'inspiration immunologique, appelées systèmes immunitaires artificiels (AIS) pour la prédiction des défauts logiciels (SFP), et plus précisément l'étude des modèles SFP inter-projets. Ce scénario de recherche manque cruellement d'intérêt par la communauté, sans parler de la rareté des travaux concernant l'utilisation des AIS pour la prédiction de défauts logiciels.

Dans notre contribution principale [HAO 20], nous avons effectué une série d'expérimentations, où nous avons utilisé 41 bases de données publiquement accessibles dans le référentiel de données PROMISE. Nos expérimentations ont impliqué 15 algorithmes d'apprentissage, dont 8 AIS. De plus, nous avons employé des tests statistiques pour obtenir des conclusions fiables et robustes sur le plan empirique. Ces tests non paramétriques, conseillés par Demsar [DEM 06],sont le test de Friedman et le test post-hoc de Nemenyi.

En effet, se contenter que de la valeur moyenne ou médiane des mesures de performances pour évaluer les algorithmes de classifications en présence de plusieurs bases de données n'est pas fiable et des tests statistiques son nécessaire pour vérifier si la différence est significative et fruit du hasard lors d'une comparaison [ARC 11].

L'analyse a été effectuée en appliquant trois métriques de performances qui sont la Précision (PR), le Rappel (PD) et la f-mesure F1. Également, nous avons conduit deux autres comparaisons, la première, compare les modèles de prédiction SFP inter-projets (PV) construits par les 15 algorithmes sélectionnés avec leurs homologues du schéma Intra-projets

(CV) par le test statistique de Wilcoxon. Enfin, la dernière comparaison évalue les rendements des modèles PV à un critère de performance proposé par He et al [HE 12], où les valeurs de Rappel et Précision doivent dépasser 70% et 50% respectivement, pour que le modèle soit déclaré acceptable.

Les résultats obtenus sont résumés ci-dessous :

- Les réponses aux questions de recherche (section 4.3) montrent que les systèmes AIS peuvent parfaitement être utilisés pour construire des modèles de prédiction inter-projets, et même être une bonne alternative à la technique SVM, spécialement pour le temps d'exécution ou pour trouver la bonne valeur des paramètres utilisateur permettant d'atteindre le meilleur résultat possible.
- Décider quelle méthode AIS est plus adaptée par rapport à une autre dépend des préférences du chef de projet. Si le but est de prédire efficacement les modules logiciels qui seront sujets aux défauts, un modèle avec un score haut de la mesure Rappel est souhaitable. Immunos-99 s'avère le plus intéressant dans ce cas même si la première version de la même technique a de meilleurs résultats, mais l'absence de paramètres pour l'ajuster rend impossible l'amélioration de ses performances. Si le modèle à une valeur de précision assez grande, ça voudrait dire que l'effort de test logiciel sera moins chargé. Dans ce cas, AIRS2.parallèle a le rendement le plus efficace. Par contre, nous ne recommandons pas l'utilisation de cette méthode à cause des résultats inconsistants que nous avons obtenus lors de nos expérimentations, il serait préférable d'utiliser AIRS2 ou CSCA. Si le chef de projet cherche un compromis entre le rappel et la précision, un score élevé de la mesure F1 est nécessaire. Immunos-99 est encore une le bon choix pour s'acquitter de cette tâche. Par contre, le test post-hoc n'a pas été concluant pour cette métrique alors nous ne pouvons pas nous prononcer avec certitude sur la validité de ce résultat.
- En évaluant les performances des systèmes AIS et en les comparant à 7 algorithmes ML référencés dans le domaine, le test statistique des résultats par la mesure Précision montre qu'il y a une différence significative entre les 15 méthodes. Seulement 37 des 105 paires comparées sont concluantes, suggérant que les performances des AIS sont assez compétitives avec les autres algorithmes même si les valeurs moyennes de la métrique de performance sont plus basses que les techniques benchmark. Par contre, l'analyse post-hoc montre qu'il n'y pas de différence, statistiquement parlant, entre les meilleurs algorithmes AIS (AIRS2, AIRS2.parallèle et CSCA) et le meilleur algorithme non immunitaires (LR) selon la mesure PR. Si on opte pour la métrique Rappel, seules 21 paires comparées sont concluantes pour rejeter l'hypothèse nulle. Nos résultats montrent qu'Immunos-1 surpasse 13 des 15 algorithmes sélectionnés après l'analyse post-hoc. Mais pour des raisons déjà mentionnées, Immunos-99 reste un choix plus judicieux pour le meilleur algorithme à utiliser afin de prédire les défauts des logiciels critiques lors du cycle de développement.

- Le temps d'exécution pourrait être un facteur adéquat pour choisir le bon algorithme pour construire des modèles de prédiction inter-projets. Nous avons implémenté un simple outil pour construire automatiquement et valider les 30 modèles pour chaque algorithme. Cela a pris 7 heures à l'algorithme SVM pour tout terminer. Alors qu'AIRS2 a pris 11 min seulement, tandis que MLP et CSCA ont dépassé les 30 minutes.
- Nous avons aussi comparé les performances des modèles intra-projets (CV) et inter-projets (PV) construits par les méthodes ML. L'analyse est réalisée par le test de Wilcoxon. Même si les valeurs moyennes des mesures de performances des modèles CV sont un peu meilleur que PV. Le test statistique n'est pas toujours concluant suggérant que les performances inter-projets sont similaires à celles produites par CV. Donc, à l'avenir, nous recommandons que les travaux futurs étudiant les modèles SFP intra-projets doivent prendre en considération l'analyse des modèles PV pour arriver à une conclusion plus générale.
- La dernière série d'expérimentations évalue le comportement des méthodes ML pour édifier des modèles de prédiction de défauts inter-projets par rapport un critère de performance proposé par He et al [HE 12]. Immunos-1 et 99 sont les meilleurs systèmes à l'égard de cette analyse, mais seulement 8 sur 30 modèles ont dépassé ce seuil pour Immunos-1 et juste 7 pour Immunos-99 montrant à quel point nous sommes loin d'atteindre les performances souhaitées, et ce, pour tous les algorithmes. Par contre, si on ne considère qu'une seule mesure disons le Rappel, Immunos-1 et Immunos-99 sont arrivés à produire des modèles remplissant le contrat par rapport à cette métrique où 22 et 24 modèles PV sont prélevés respectivement. 19 sur 30 est la meilleure note pour le seuil de la mesure PR, résultant des modèles construits par Naive Bayes et la régression logistique. Trois des systèmes AIS ont atteint 15 (AIRS2, AIRS2.parallèle et CSCA). Cette observation démontre l'importance du choix de l'algorithme approprié pour avoir un processus de prédiction de défauts efficace.
- Enfin, l'algorithme Immunos-2 ne semble pas adapté pour la prédiction de défauts logiciels inter-projets selon nos 3 benchmarks.

Dans nos futurs travaux, nous projetons d'analyser en profondeur l'algorithme Immunos-81 et surtout sa version hybride Immunos-99. Nous avons aussi entrepris une étude intensive des systèmes AIS dans tous les scénarios possibles de prédiction de défauts logiciels. Cette étude fera l'objet de publications futures.

D'autre part, un autre type de méthodes bio-inspirées ont suscité notre intérêt pendant notre travail de recherche car elles semblent bien adaptées pour la prédiction de défauts logiciels. Il s'agit des méthodes issues de l'intelligence en essaim (Swarm intelligence) telles que celle qui simule la méthode de chasse des loups gris (gray wolf optimization algorithm). Nous projetons de prospecter ce genre d'approches dans nos prochaines études pour la construction de classifieurs destinés à la prédiction de défauts logiciels.

Bibliographie

ABA 14	G. Abaei and A. Selamat, “A survey on software fault detection based on different prediction approaches,” *Vietnam J. Comput. Sci.*, vol. 1, no. 2, pp. 79–95, 2014.
ABA 16	M. Abadi *et al.*, “TensorFlow: A system for large-scale machine learning,” in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016*, 2016, pp. 265–283.
ABD 15	Y. Abdi, S. Parsa, and Y. Seyfari, “A hybrid one-class rule learning approach based on swarm intelligence for software fault prediction,” *Innov. Syst. Softw. Eng.*, vol. 11, no. 4, pp. 289–301, 2015.
ABR 10	A. Abran, *Software Metrics and Software Metrology*. John Wiley \& Sons, 2010.
AKO 17	M. Akour, I. Alsmadi, and I. Alazzam, “Software fault proneness prediction: A comparative study between bagging, boosting, and stacking ensemble and base learner methods,” *Int. J. Data Anal. Tech. Strateg.*, vol. 9, no. 1, pp. 1–16, 2017.
AL 10	R. E. Al-Qutaish, “Quality Models in Software Engineering Literature: An Analytical and Comparative Study,” *J. Am. Sci.*, vol. 6, no. 3, pp. 166–175, 2010.
AL 11	H. A. Al-Jamimi and L. Ghouti, “Efficient prediction of software fault proneness modules using support vector machines and probabilistic neural networks,” in *2011 5th Malaysian Conference in Software Engineering, MySEC 2011*, 2011, pp. 251–256.
ALC 09	J. Alcalá-Fdez, L. Sánchez, S. García, M.J. del Jesus, S. Ventura, J.M. Garrell, J. Otero, C. Romero, J. Bacardit, V.M. Rivas, J.C. Fernández, F. Herrera, “KEEL: A software tool to assess evolutionary algorithms for data mining problems,” Soft Comput., vol. 13, no. 3, pp. 307–318, 2009.
ALC 11	J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera., “KEEL data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework,” J. Mult. Log. Soft Comput., vol. 17, no. 2–3, pp. 255–287, 2011.
ALG 20	H. Alsghaier and M. Akour, “Software fault prediction using particle swarm algorithm with genetic algorithm and support vector machine classifier,” *Softw. Pract. Exp.*, vol. 50, no. 4, pp. 407–427, 2020.
ALI 20	A. Ali and C. Gravino, “Bio-inspired Algorithms in Software Fault Prediction: A Systematic Literature Review,” in *2020 14th International Conference on Open Source Systems and Technologies, ICOSST 2020 - Proceedings*, 2020, pp. 1–8.
ALP 14	E. Alpaydin, *Introduction to machine learning*, Third Edition.. 2014.
ALQ 20	O. Al Qasem, M. Akour, and M. Alenezi, “The Influence of Deep Learning Algorithms Factors in Software Fault Prediction,” *IEEE Access*, vol. 8, pp. 63945–63960, 2020.
ALS 20	H. Alsolai and M. Roper, “A systematic literature review of machine learning techniques for software maintainability prediction,” *Inf. Softw. Technol.*, vol. 119, p. 106214, 2020.
ALT 17	H. Altinger, S. Herbold, F. Schneemann, J. Grabowski, and F. Wotawa, “Performance tuning for automotive Software Fault Prediction,” in *SANER 2017 - 24th IEEE International Conference on Software Analysis, Evolution, and Reengineering*, 2017, pp. 526–530.
AME 19	S. Amershi *et al.*, “Software Engineering for Machine Learning: A Case Study,” in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, 2019, pp. 291–300.

ARC 11	A. Arcuri and L. Briand, “A practical guide for using statistical tests to assess randomized algorithms in software engineering,” in *Proceedings - International Conference on Software Engineering*, 2011, pp. 1–10.
BAN 02	J. Bansiya and C. G. Davis, “A hierarchical model for object-oriented design quality assessment,” *IEEE Trans. Softw. Eng.*, vol. 28, no. 1, pp. 4–17, 2002.
BAS 96	V. R. Basili, L. C. Briand, and W. L. Melo, “A validation of object-oriented design metrics as quality indicators,” *IEEE Trans. Softw. Eng.*, vol. 22, no. 10, pp. 751–761, 1996.
BEE 10	S. Beecham, T. Hall, D. Bowes, D. Gray, S. Counsell, and S. Black, “A systematic review of fault prediction approaches used in software engineering,” *Irish Softw. Eng. Res. Cent. Limerick, Irel.*, 2010.
BEI 03	B. Beizer, *Software testing techniques*. Dreamtech Press, 2003.
BER 91	.H. Bersini and F. J. Varela, “Hints for adaptive problem solving gleaned from immune networks,” in *Parallel Problem Solving from Nature, Lecture Notes in Computer Science Volume 496*, S. H.-P. & M. R., Ed. Berlin: Springer Verlag, 1991, pp. 343–354.
BIS 06	C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.
BIS 11	P. S. Bishnu and V. Bhattacherjee, “Software fault prediction using quad tree-based K-means clustering algorithm,” *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 6, pp. 1146–1150, 2011.
BOE 78	B. W. Boehm, J. R. Brown, and M. Lipow, “Quantitative evaluation of software quality,” in *Proceedings - International Conference on Software Engineering*, 1978, pp. 592–605.
BRI 00	L. C. Briand, J. Wüst, J. W. Daly, and D. V. Porter, “Exploring the relationships between design measures and software quality in object-oriented systems,” *J. Syst. Softw.*, vol. 51, no. 3, pp. 245–273, 2000.
BRI 01	L. C. Briand, J. Wüst, and H. Lounis, “Replicated case studies for investigating quality factors in object-oriented designs,” *Empir. Softw. Eng.*, vol. 6, no. 1, pp. 11–58, 2001.
BRI 99	L. C. Briand, J. Wuest, S. V. Ikonomovski, and H. Lounis, “Investigating quality factors in object-oriented designs: An industrial case study,” in *Proceedings - International Conference on Software Engineering*, 1999, pp. 345–354.
BRO 05	J. Brownlee, “Artificial immune recognition system (airs)-a review and analysis”, *Cent. Intell. Syst. Complex Process. (CISCP), Fac. Inf. Commun. Technol. (ICT), Swinburne Univ. Technol.*, Victoria, Australia, 2005.
BRO 05 A	J. Brownlee, “Clonal selection theory & clonalg-the clonal selection classification algorithm (CSCA),” *Cent. Intell. Syst. Complex Process. (CISCP), Fac. Inf. Commun. Technol. (ICT), Swinburne Univ. Technol.*, Victoria, Australia, 2005.
BRO 05 B	J. Brownlee, “Immunos-81, the misunderstood artificial immune system,” *Cent. Intell. Syst. Complex Process. (CISCP), Fac. Inf. Commun. Technol. (ICT), Swinburne Univ. Technol.*, Victoria, Australia, 2005.
BRO 11	J. Brownlee, *Clever Algorithms - Nature-Inspired Programming Recipes*, 1st ed. Lulu.com, 2011.
BRO 87	F. P. Brooks, “Essence and Accidents of Software Engineering,” *Computer (Long. Beach. Calif).*, vol. 20, no. 4, pp. 10–19, 1987.
BUR 11	F. M. Burnet, “The clonal selection theory of acquired immunity.,” *clonal Sel. theory Acquir. immunity.*, pp. 275–294, 2011.
BUR 20	A. Burkov, *Machine Learning Engineering*. True Positive Incorporated, 2020
BUR 57	F. M. Burnet, “A modification of Jerne’s theory of antibody production using the concept of

	clonal selection.," *Aust. J. Sci.*, vol. 20, no. 3, pp. 67–69, 1957.
CAN 13	G. Canfora, A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Multi-objective cross-project defect prediction," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, 2013, pp. 252–261.
CAR 00	J. H. Carter, "The immune system as a model for pattern recognition and classification," *J. Am. Med. Informatics Assoc.*, vol. 7, no. 1, pp. 28–41, 2000.
CAS 02	L. N. Castro, L. N. De Castro, and J. Timmis, *Artificial immune systems: a new computational intelligence approach*. Springer Science & Business Media, 2002.
CAT 07	C. Catal and B. Diri, "Software defect prediction using artificial immune recognition system," in *Proceedings of the IASTED International Conference on Software Engineering, SE 2007*, Anaheim, CA, USA: ACTA Press.2007, pp. 285–290..
CAT 07 A	C. Catal and B. Diri, "Software fault prediction with object-oriented metrics based artificial immune recognition system," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2007, vol. 4589 LNCS, pp. 300–314.
CAT 09	C. Catal and B. Diri, "A systematic review of software fault prediction studies," *Expert Syst. Appl.*, vol. 36, no. 4, pp. 7346–7354, 2009.
CAT 09 A	C. Catal, U. Sevim, and B. Diri, "Clustering and metrics thresholds based software fault prediction of unlabeled program modules," in *ITNG 2009 - 6th International Conference on Information Technology: New Generations*, 2009, pp. 199–204.
CAT 09 B	C. Catal and B. Diri, "Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem," *Inf. Sci. (Ny).*, vol. 179, no. 8, pp. 1040–1058, 2009.
CAT 11	C. Catal, "Software fault prediction: A literature review and current trends," *Expert Syst. Appl.*, vol. 38, no. 4, pp. 4626–4636, 2011.
CAT 11 A	C. Catal, U. Sevim, and B. Diri, "Practical development of an Eclipse-based software fault prediction tool using Naive Bayes algorithm," *Expert Syst. Appl.*, vol. 38, no. 3, pp. 2347–2353, 2011.
CAT 12	C. Catal, "Performance evaluation metrics for software fault prediction studies," *Acta Polytech. Hungarica*, vol. 9, no. 4, pp. 193–206, 2012.
CHI 91	S. R. Chidamber and C. F. Kemerer, "Towards a metrics suite for object oriented design," in *ACM SIGPLAN Notices*, 1991, vol. 26, no. 11, pp. 197–211.
CHI 94	S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476–493, 1994.
CHO 18	G. R. Choudhary, S. Kumar, K. Kumar, A. Mishra, and C. Catal, "Empirical analysis of change metrics for software fault prediction," *Comput. Electr. Eng.*, vol. 67, pp. 15–24, 2018.
COR 11	A. Cornuéjols and L. Miclet, *Apprentissage artificiel: concepts et algorithmes*. Editions Eyrolles, 2011.
CRU 09	A. E. C. Cruz and K. Ochimizu, "Towards logistic regression models for predicting fault-prone code across software projects," in *2009 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM 2009*, 2009, pp. 460–463.
CUT 02	V. Cutello and G. Nicosia, "An immunological approach to combinatorial optimization problems," in *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*, 2002, vol. 2527, pp. 361–370.

CUT 05	V. Cutello and G. Nicosia, "The clonal selection principle for in silico and in vitro computing," in *Recent developments in biologically inspired computing*, Igi Global, 2005, pp. 140–147.
DAM 09	M. D'Ambros, M. Lanza, and R. Robbes, "On the relationship between change coupling and software defects," in *Proceedings - Working Conference on Reverse Engineering, WCRE*, 2009, pp. 135–144.
DAS 09	D. Dasgupta and F. Nino, *Immunological computation: theory and applications*. CRC press, 2009.
DEC 01	L. N. De Castro and F. J. Von Zuben, "aiNet: an artificial immune network for data analysis," in *Data Mining: a heuristic approach*, vol. 1, IGI Global, 2001, pp. 231–259.
DEC 02	L. N. De Castro and F. J. Von Zuben, "Learning and optimization using the clonal selection principle," *IEEE Trans. Evol. Comput.*, vol. 6, no. 3, pp. 239–251, 2002.
DEC 02 B	L. N. De Castro and J. Timmis, "An artificial immune network for multimodal function optimization," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, 2002, vol. 1, pp. 699–704.
DEC 99	F. J. V. Z. de Castro, "Artificial immune systems. Part I. Basic theory and applications," *Univ. Estadual Campinas, Dezembro de, Tech. Rep*, vol. 210, no. 1, 1999.
DEI 09	F. Deissenboeck, E. Juergens, K. Lochmann, and S. Wagner, "Software quality models: Purposes, usage scenarios and requirements," in *Proceedings - International Conference on Software Engineering*, 2009, pp. 9–14.
DEM 06	J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, no. Jan, pp. 1–30, 2006.
DHA 96	P. D'haeseleer, S. Forrest, and P. Helman, "Immunological approach to change detection: algorithms, analysis and implications," in *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, 1996, pp. 110–119.
DOS 21	M. R. Dos Santos Barcelos, C. F. Simões Gomes, A. Manzolillo Sanseverino, and M. Dos Santos, "Literature review on software metrics and a New Proposal," *Exatas Eng.*, vol. 11, no. 32, pp. 33–59, 2021.
DRO 95	R. G. Dromey, "A model for software product quality," *IEEE Trans. Softw. Eng.*, vol. 21, no. 2, pp. 146–162, 1995.
EL 01	K. El Emam, W. Melo, and J. C. Machado, "The prediction of faulty classes using object-oriented design metrics," *J. Syst. Softw.*, vol. 56, no. 1, pp. 63–75, 2001.
EL 01 A	K. El Emam, S. Benlarbi, N. Goel, and S. N. Rai, "The confounding effect of class size on the validity of object-oriented metrics," *IEEE Trans. Softw. Eng.*, vol. 27, no. 7, pp. 630–650, 2001.
ELI 08	K. O. Elish and M. O. Elish, "Predicting defect-prone software modules using support vector machines," *J. Syst. Softw.*, vol. 81, no. 5, pp. 649–660, 2008.
ERT 15	E. Erturk and E. A. Sezer, "A comparison of some soft computing methods for software fault prediction," *Expert Syst. Appl.*, vol. 42, no. 4, pp. 1872–1879, 2015.
FAR 86	J. D. Farmer, N. H. Packard, and A. S. Perelson, "The immune system, adaptation, and machine learning," *Phys. D Nonlinear Phenom.*, vol. 22, no. 1–3, pp. 187–204, 1986.
FEN 00	N. E. Fenton and M. Neil, "Software metrics: roadmap," in *Proceedings of the Conference on the Future of Software Engineering*, 2000, pp. 357–370.
FEN 00 A	N. E. Fenton, "Quantitative analysis of faults and failures in a complex software system," *IEEE Trans. Softw. Eng.*, vol. 26, no. 8, pp. 797–814, 2000.
FEN 97	N. Fenton and S. L. Pfleeger, *Software Metrics, A Rigorous and Practical Approach*. CRC press,

	1997.
FOR 94	S. Forrest, L. Allen, A. S. Perelson, and R. Cherukuri, "Self-nonself discrimination in a computer," *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*. pp. 202–212, 1994.
FRI 37	M. Friedman, "The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance," *J. Am. Stat. Assoc.*, vol. 32, no. 200, pp. 675–701, 1937.
GAR 04	S. M. Garrett, "Parameter-free, adaptive clonal selection," in *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)*, 2004, vol. 1, pp. 1052–1058.
GAT 20	V. Garousi, A. Rainer, P. Lauvås, and A. Arcuri, "Software-testing education: A systematic literature mapping," *J. Syst. Softw.*, vol. 165, p. 110570, 2020.
GEN 05	M. Genero, M. Piattini, and C. Calero, "A survey of metrics for UML class diagrams," *J. Object Technol.*, vol. 4, no. 9, pp. 59–92, 2005.
GHA 06	M. Gharbi, "Optimisation grâce aux Systèmes Immunitaires Artificiels," *Rapp. stage Master, Rech. Informatique, IFSIC, Rennes, Fr.*, 2006.
GLA 00	D. Glasberg, K. El Emam, W. Melo, and N. Madhavji, *Validating object-oriented design metrics on a commercial java application*. Citeseer, 2000.
GON 03	F. González, D. Dasgupta, and L. F. Niño, "A randomized real-valued negative selection algorithm," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2003, vol. 2787, pp. 261–272.
GRA 00	T. L. Graves, A. F. Karr, U. S. Marron, and H. Siy, "Predicting fault incidence using software change history," *IEEE Trans. Softw. Eng.*, vol. 26, no. 7, pp. 653–661, 2000.
GRA 92	R. B. Grady, *Practical software metrics for project management and process improvement*. Prentice-Hall, Inc., 1992.
GRE 06	J. Greensmith, U. Aickelin, and J. Twycross, "Articulation and clarification of the dendritic cell algorithm," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2006, vol. 4163 LNCS, pp. 404–417.
GRE 07	J. Greensmith, "The dendritic cell algorithm," Thèse de doctorat. University of Nottingham, UK, 2007.
GRE 09	J. Greensmith and U. Aickelin, "Artificial dendritic cells: Multi-faceted perspectives," in *Studies in Computational Intelligence*, vol. 182, Springer, 2009, pp. 375–395.
GYI 05	T. Gyimóthy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Trans. Softw. Eng.*, vol. 31, no. 10, pp. 897–910, 2005.
HAL 00	G. A. Hall and J. C. Munson, "Software evolution: Code delta and code churn," *J. Syst. Softw.*, vol. 54, no. 2, pp. 111–118, 2000.
HAL 09	M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software," *ACM SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, 2009.
HAL 12	T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A Systematic Literature Review on Fault Prediction Performance in Software Engineering," *IEEE Trans. Softw. Eng.*, vol. 38, no. 6, pp. 1276–1304, 2012.
HAL 77	M. H. Halstead, *Elements of software science library*. Elsevier Science Inc., 1977.
HAM 19	M. Hammad, A. Alqaddoumi, H. Al-Obaidy, and K. Almseidein, "Predicting software faults based on k-nearest neighbors classification," *Int. J. Comput. Digit. Syst.*, vol. 8, no. 5, pp. 461–

	467, 2019.
HAN 12	J. Han, M. Kamber, and J. Pei, “Introduction,” in *Data Mining*, Third Edition., Eds. Boston: Morgan Kaufmann, 2012, pp. 1–38.
HAN 15	D. J. Hand and N. M. Adams, “Data Mining,” in *Wiley StatsRef: Statistics Reference Online*, American Cancer Society, 2015, pp. 1–7.
HAO 17	A. T. Haouari, L. Souici-Meslati, F. Atil, “Outil d’aide à Prédiction des Défauts Logiciels,” in *La Conférence Maghrébine sur les Avancés des Systèmes Décisionnels (ASD) du 27 au 29 avril*, Tabarka, Tunisie 2017.
HAO 17 A	A. T. Haouari, L. Souici-Meslati, and F. Atil, “Systèmes Immunitaires Artificiels pour la Prédiction de Défauts Logiciels,” in *Colloque sur l’Optimisation et les Systèmes d’Information COSI’2017, 14 au 16 Mai 2017*, Bouira, Algeria, 2017.
HAO 17 B	A. T. Haouari, L. Souici-Meslati, and F. Atil, “Prédiction de défauts logiciels par des classifieurs immunologiques,” in *SFC: XXIV èmes rencontres de la société francophone de classification 28 – 30 Juin*, Lyon, France, 2017.
HAO 20	A. T. Haouari, L. Souici-Meslati, F. Atil, and D. Meslati, “Empirical comparison and evaluation of Artificial Immune Systems in inter-release software fault prediction,” *Appl. Soft Comput. J.*, vol. 96, p. 106686, 2020.
HAR 12	M. Harman, “The role of artificial intelligence in software engineering,” in *2012 1st International Workshop on Realizing AI Synergies in Software Engineering, RAISE 2012 - Proceedings*, 2012, pp. 1–6.
HAR 14	M. Harman, S. Islam, Y. Jia, L. L. Minku, F. Sarro, and K. Srivisut, “Less is more: Temporal fault predictive performance over multiple hadoop releases,” in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2014, vol. 8636 LNCS, pp. 240–246.
HAS 21	Y. Hassouneh, H. Turabieh, T. Thaher, I. Tumar, H. Chantar, and J. Too, “Boosted Whale Optimization Algorithm with Natural Selection Operators for Software Fault Prediction,” *IEEE Access*, vol. 9, pp. 14239–14258, 2021.
HE 12	Z. He, F. Shu, Y. Yang, M. Li, and Q. Wang, “An investigation on the feasibility of cross-project defect prediction,” *Autom. Softw. Eng.*, vol. 19, no. 2, pp. 167–199, 2012.
HE 15	P. He, B. Li, X. Liu, J. Chen, and Y. Ma, “An empirical study on software defect prediction with a simplified metric set,” *Inf. Softw. Technol.*, vol. 59, pp. 170–190, 2015.
HEN 96	Henderson and Sellers, *Object-Oriented Metrics: measures of Complexity*. USA: Prentice-Hall, Inc., 1996.
HER 16	S. Herbold, A. Trautsch, and J. Grabowski, “Global vs. local models for cross-project defect prediction: A replication study,” *Empir. Softw. Eng.*, vol. 22, no. 4, pp. 1866–1902, 2016.
HER 18	S. Herbold, A. Trautsch, and J. Grabowski, “A Comparative Study to Benchmark Cross-Project Defect Prediction Approaches,” *IEEE Trans. Softw. Eng.*, vol. 44, no. 9, pp. 811–833, 2018.
HOF 99	S. F. S. Hofmeyr, “Immunity by design: an artificial immune system,” in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 99)*, 1999, p. 1289-1296.
HOL 94	G. Holmes, A. Donkin, and I. H. Witten, “WEKA: A machine learning workbench,” in *Australian and New Zealand Conference on Intelligent Information Systems - Proceedings*, 1994, pp. 357–361.
HON 12	E. Hong, “Software fault-proneness prediction using random forest,” *Int. J. Smart Home*, vol. 6, no. 4, pp. 147–152, 2012.

IEE 06	IEEE,"ISO/IEC/IEEE International Standard for Software Engineering - Software Life Cycle Processes - Maintenance," *ISO/IEC 147642006 IEEE Std 14764-2006 Revis. IEEE Std 1219-1998)*, pp. 1–58, 2006.
IEE 90	IEEE, "IEEE Standard Glossary of Software Engineering Terminology," *IEEE Std 610.12-1990*, pp. 1–84, 1990.
IEE 91	IEEE, "IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries," *IEEE Std 610*, pp. 1–217, 1991.
ILL 10	T. Illes-Seifert and B. Paech, "Exploring the relationship of a file's history and its fault-proneness: An empirical method and its application to open source programs," *Inf. Softw. Technol.*, vol. 52, no. 5, pp. 539–558, 2010.
ISO 01	International Organization for Standardization (ISO), "ISO/IEC 9126--Software Engineering--Product Quality." Geneve, 2001.
ISO 11	International Organization for Standardization (ISO), "ISO/IEC 25010:2011, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models." 2011.
JAL 09	P. Jalote, *A concise introduction to software engineering*, vol. 46, no. 06. Springer Science & Business Media, 2009.
JAN 12	T. Jansen and C. Zarges, "Artificial immune systems for optimisation," in *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*, 2012, pp. 1059–1078.
JER 74	N. K. Jerne, "Towards a network theory of the immune system," *Collect.Ann.Inst.Pasteur*, vol. 125 C, no. 1–2, , 1974, pp. 373–389
JER 74 A	N. K. Jerne, "Clonal selection in a lymphocyte network.," *Soc. Gen. Physiol. Ser.*, vol. 29, 1974, pp. 39–48
JI 04	Z. Ji and D. Dasgupata, "Augmented negative selection algorithm with variable-coverage detectors," in *Proceedings of the 2004 Congress on Evolutionary Computation, CEC2004*, 2004, vol. 1, pp. 1081–1088.
JI 05	Z. Ji, "A boundary-aware negative selection algorithm," in *Proceedings of the 9th IASTED International Conference on Artificial Intelligence and Soft Computing, ASC 2005*, 2005, pp. 340–345.
JIA 08	Y. Jiang, B. Cukic, and Y. Ma, "Techniques for evaluating fault prediction models," *Empir. Softw. Eng.*, vol. 13, no. 5, pp. 561–595, 2008.
JIN 09	Z. Jinquan, L. Xiaojie, L. Tao, L. Caiming, P. Lingxi, and S. Feixian, "A self-adaptive negative selection algorithm used for anomaly detection," *Prog. Nat. Sci.*, vol. 19, no. 2, pp. 261–266, 2009.
JUN 19	K. Juneja, "A fuzzy-filtered neuro-fuzzy framework for software fault prediction for inter-version and inter-project evaluation," *Appl. Soft Comput. J.*, vol. 77, pp. 696–713, 2019.
JOR 95	M. Jorgensen, "Experience with the accuracy of software maintenance task effort prediction models," *IEEE Trans. Softw. Eng.*, vol. 21, no. 8, pp. 674–681, 1995.
JUR 10	M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *ACM International Conference Proceeding Series*, 2010, pp. 1–10.
JUR 10 A	M. Jureczko and D. Spinellis, "Using Object-Oriented Design Metrics to Predict Software Defects," *Model. Methods Syst. Dependability. Oficyna Wydawnicza Politech. Wrocławskiej*, pp. 69–81, 2010.
KAN 03	S. H. Kan, *Metrics and models in software quality engineering*. Addison-Wesley Professional,

	2003.
KAN 07	S. Kanmani, V. R. Uthariaraj, V. Sankaranarayanan, and P. Thambidurai, "Object-oriented software fault prediction using neural networks," *Inf. Softw. Technol.*, vol. 49, no. 5, pp. 483–492, 2007.
KAU 08	A. Kaur and R. Malhotra, "Application of random forest in predicting fault-prone classes," in *Proceedings - 2008 International Conference on Advanced Computer Theory and Engineering, ICACTE 2008*, 2008, pp. 37–43.
KAU 15	A. Kaur and K. Kaur, "An empirical study of robustness and stability of machine learning classifiers in software defect prediction," in Advances in Intelligent Systems and Computing, vol. 320, E.-S. M. El-Alfy, S. M. Thampi, H. Takagi, S. Piramuthu, and T. Hanne, Eds. Cham: Springer International Publishing, 2015, pp. 383–397.
KAU 16	A. Kaur and K. Kaur, "Micro-interaction metrics based software defect prediction with machine learning, immune inspired and evolutionary classifiers: An empirical study," in *Smart Innovation, Systems and Technologies*, 2016, vol. 50, pp. 221–233.
KHA 21	Y. Khatri and S. K. Singh, "Cross project defect prediction: a comprehensive survey with its SWOT analysis," *Innov. Syst. Softw. Eng.*, pp. 1–19, 2021.
KHO 00	T. M. Khoshgoftaar, R. Shan, and E. B. Allen, "Using product, process, and execution metrics to predict fault-prone software modules with classification trees," in Proceedings of IEEE International Symposium on High Assurance Systems Engineering, 2000, vol. 2000-January, pp. 301–310.
KHO 01	T. M. Khoshgoftaar and E. B. Allen, "Empirical Assessment of a Software Metric: The Information Content of Operators," *Softw. Qual. J.*, vol. 9, no. 2, pp. 99–112, 2001.
KHO 03	T. M. Khoshgoftaar and N. Seliya, "Analogy-Based Practical Classification Rules for Software Quality Estimation," *Empir. Softw. Eng.*, vol. 8, no. 4, pp. 325–350, 2003.
KIT 10	B. Kitchenham, "What's up with software metrics? - A preliminary mapping study," *J. Syst. Softw.*, vol. 83, no. 1, pp. 37–51, 2010.
KIT 96	B. Kitchenham and S. L. Pfleeger, "Software quality: the elusive target," *IEEE Softw.*, vol. 13, no. 1, pp. 12–21, 1996.
KLE 06	J. Kleinberg and E. Tardos, *Algorithm design*. Pearson Education India, 2006.
KNI 01	T. Knight and J. Timmis, "AINE: An immunological approach to data mining," in *Proceedings - IEEE International Conference on Data Mining, ICDM*, 2001, pp. 297–304.
KOP 06	H. Kopackova and L. Kopacek, "Text-based decision making with artificial immune systems.," *WSEAS Trans. Bus. Econ.*, vol. 3, no. 5, pp. 428–433, 2006.
KOR 08	A. G. Koru, K. El Emam, D. Zhang, H. Liu, and D. Mathew, "Theory of relative defect proneness," *Empir. Softw. Eng.*, vol. 13, no. 5, pp. 473–498, 2008.
KOR 09	A. G. Koru, D. Zhang, K. El Emam, and H. Liu, "An investigation into the functional form of the size-defect relationship for software modules," *IEEE Trans. Softw. Eng.*, vol. 35, no. 2 SPEC. ISS., pp. 293–304, 2009.
KUM 16	P. Kumudha and R. Venkatesan, "Cost-Sensitive Radial Basis Function Neural Network Classifier for Software Defect Prediction," *Sci. World J.*, vol. 2016, 2016.
KUM 18	S. Kumar and S. S. Rathore, *Software Fault Prediction: A Road Map*. Springer, 2018
KUM 18 A	K. K. B, J. Gyani, and G. Narsimha, "Software Defect Prediction using Ant Colony Optimization," *Int. J. Appl. Eng. Res.*, vol. 13, no. 19, pp. 14291–14297, 2018.
LAM 11	A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonckz, "Comparing mining algorithms for

	predicting the severity of a reported bug," in *Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR*, 2011, pp. 249–258.
LAY 08	L. Layman, G. Kudrjavets, and N. Nagappan, "Iterative identification of fault-prone binaries using inprocess metrics," in ESEM'08: Proceedings of the 2008 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, 2008, pp. 206–212.
LEE 11	T. Lee, J. Nam, D. Han, S. Kim, and H. Peter In, "Developer Micro Interaction Metrics for Software Defect Prediction," in *IEEE Transactions on Software Engineering*, 2016, vol. 42, no. 11, pp. 1015–1035.
LES 08	S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Trans. Softw. Eng.*, vol. 34, no. 4, pp. 485–496, 2008.
LI 93	W. Li and S. Henry, "Object-oriented metrics that predict maintainability," *J. Syst. Softw.*, vol. 23, no. 2, pp. 111–122, 1993.
LIN 08	R. Lincke, J. Lundberg, and W. Löwe, "Comparing software metrics tools," in *ISSTA'08: Proceedings of the 2008 International Symposium on Software Testing and Analysis 2008*, 2008, pp. 131–141.
MAD 15	L. Madeyski and M. Jureczko, "Which process metrics can significantly improve defect prediction models? An empirical study," *Softw. Qual. J.*, vol. 23, no. 3, pp. 393–422, Sep. 2015.
MAL 15	R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Appl. Soft Comput. J.*, vol. 27, pp. 504–518, 2015.
MAL 16	R. Malhotra, "An empirical framework for defect prediction using machine learning techniques with Android software," *Appl. Soft Comput. J.*, vol. 49, pp. 1034–1050, 2016.
MAN 18	C. Manjula and L. Florence, "Deep neural network based hybrid approach for software defect prediction using software metrics," *Cluster Comput.*, vol. 22, no. 4, pp. 9847–9863, 2018.
MAN 47	H. B. Mann and D. R. Whitney, "On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other," *Ann. Math. Stat.*, vol. 18, no. 1, pp. 50–60, 1947.
MAR 94	R. Martin, "OO design quality metrics," *An Anal. Depend.*, vol. 12, pp. 151–170, 1994.
MAT 02	P. Matzinger, "The danger model: A renewed sense of self," *Science (80-.).*, vol. 296, no. 5566, pp. 301–305, 2002.
MAT 10	S. Matsumoto, Y. Kamei, A. Monden, K. I. Matsumoto, and M. Nakamura, "An analysis of developer metrics for fault prediction," in *ACM International Conference Proceeding Series*, 2010, pp. 1–9.
MAT 94	P. Matzinger, "Tolerance, danger, and the extended family," *Annu. Rev. Immunol.*, vol. 12, no. 1, pp. 991–1045, 1994.
MCC 58	J. McCarthy, "Programs with Common Sense," in *Proceedings of the Teddington Conference on the Mechanization of Thought Processes,756-91. London: Her Majesty's Stationery Office*, 1958, pp. 77–84.
MCC 76	T. J. McCabe, "A Complexity Measure," *IEEE Trans. Softw. Eng.*, vol. SE-2, no. 4, pp. 308–320, 1976.
MCC 77	J. a. McCall, P. K. Richards, and G. F. Walters, "Factors in Software Quality," *at'l Tech. Inf. Serv.*, vol. 1, 2 and 3, no. ADA049055, pp. 689–1699, 1977.
MEN 02	T. Menzies, J. S. Di Stefano, M. Chapman, and K. McGill, "Metrics that matter," in *27th Annual NASA Goddard/IEEE Software Engineering Workshop, 2002. Proceedings.*, 2002, pp. 51–57.
MEN 04	T. Menzies and J. S. Di Stefano, "How good is your blind spot sampling policy?," in

	Proceedings of IEEE International Symposium on High Assurance Systems Engineering, 2004, vol. 8, pp. 129–138.
MEN 07	T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 2–13, 2007.
MEN 10	T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: Current results, limitations, new approaches," *Autom. Softw. Eng.*, vol. 17, no. 4, pp. 375–407, 2010.
MEN 16	T. Menzies, R. Krishna, and D. Pryor, "The promise repository of empirical software engineering data. North Carolina State University." 2016. http://openscience.us/repo
MIC 05	J. Michura and M. A. M. Capretz, "Metrics suite for class complexity," in *International Conference on Information Technology: Coding and Computing, ITCC*, 2005, vol. 2, pp. 404–409.
MIG 14	J. P. Miguel, D. Mauricio, and G. Rodríguez, "A Review of Software Quality Models for the Evaluation of Software Products," *Int. J. Softw. Eng. Appl.*, vol. 5, no. 6, pp. 31–53, 2014.
MOL 03	K. Molokken and M. Jorgensen, "A review of software surveys on software effort estimation," in *2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings.*, 2003, pp. 223–230.
MON 13	A. Monden *et al.*, "Assessing the cost effectiveness of fault prediction in acceptance testing," *IEEE Trans. Softw. Eng.*, vol. 39, no. 10, pp. 1345–1357, Oct. 2013.
MOR 01	S. Morasca, "Software measurement," in *Handbook of Software Engineering and Knowledge Engineering: Volume I: Fundamentals*, World Scientific, 2001, pp. 239–276.
MOS 08	R. Moser, W. Pedrycz, and G. Succi, "A Comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Proceedings - International Conference on Software Engineering*, 2008, pp. 181–190.
MUN 98	J. C. Munson and S. G. Elbaum, "Code churn: a measure for estimating the impact of code change," in *Conference on Software Maintenance*, 1998, pp. 24–31.
MYE 08	G. J. Myers, T. Badgett, T. M. Thomas, and C. Sandler, *The art of software testing*, vol. 3. Wiley Online Library, 2008.
NAI 08	K. Naik and P. Tripathy, *Software Testing and Quality Assurance: Theory and Practice*. John Wiley & Sons, 2008.
NAT 68	N. S. Committee, "Proceedings of the NATO Conference on Software Engineering,", Garmisch, Germany, 1968.
NEM 63	P. B. Nemenyi, *Distribution-free multiple comparisons.* Princeton University, 1963.
NEU 92	A. U. Neumann and G. Weisbuch, "Dynamics and topology of idiotypic networks," *Bull. Math. Biol.*, vol. 54, no. 5, pp. 699–726, 1992.
NIE 15	M. A. Nielsen, *Neural networks and deep learning*, vol. 25. Determination press San Francisco, CA, 2015.
OLA 07	H. M. Olague, L. H. Etzkorn, S. Gholston, and S. Quattlebaum, "Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly Iterative or agile software development processes," IEEE Trans. Softw. Eng., vol. 33, no. 6, pp. 402–419, 2007.
OLA 08	H. M. Olague, L. H. Etzkorn, S. L. Messimer, and H. S. Delugach, "An empirical validation of object-oriented class complexity metrics and their ability to predict error-prone classes in highly iterative, or agile, software: A case study," J. Softw. Maint. Evol., vol. 20, no. 3, pp. 171–197,

	2008.
OLI 12	L. O. V. B. Oliveira, R. L. M. Mota, and D. A. C. Barone, "Clonal Selection Classifier with Data Reduction: Classification as an optimization task," in *2012 IEEE Congress on Evolutionary Computation, CEC 2012*, 2012, pp. 1–7.
OST 05	T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the location and number of faults in large software systems," *IEEE Trans. Softw. Eng.*, vol. 31, no. 4, pp. 340–355, 2005.
OST 10	T. Ostrand and E. Weyuker, "Software fault prediction tool," in *ISSTA'10 - Proceedings of the 2010 International Symposium on Software Testing and Analysis*, 2010, pp. 275–278.
PAI 07	G. J. Pai and J. B. Dugan, "Empirical analysis of software fault content and fault proneness using Bayesian methods," *IEEE Trans. Softw. Eng.*, vol. 33, no. 10, pp. 675–686, 2007.
PAN 21	S. K. Pandey, R. B. Mishra, and A. K. Tripathi, "Machine learning based methods for software fault prediction: A survey," *Expert Syst. Appl.*, p. 114595, 2021.
PRO 15	Proceedings of the 37th International Conference on Software Engineering, "ICSE '15: Proceedings of the 37th International Conference on Software Engineering - Volume 1," , Firenze, Italy,2015.
RAD 13	D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič, "Software fault prediction metrics: A systematic literature review," *Inf. Softw. Technol.*, vol. 55, no. 8, pp. 1397–1418, 2013.
RAJ 18	K. S. Raju, M. R. Murty, M. V. Rao, and S. C. Satapathy, "Support Vector Machine with K-fold Cross Validation Model for Software Fault Prediction," *Int. J. Pure Appl. Math.*, vol. 118, no. 20, pp. 321–334, 2018.
RAM 14	R. Ramler, J. Himmelbauer, and T. Natschläger, "Building defect prediction models in practice," in *Computer Systems and Software Engineering: Concepts, Methodologies, Tools, and Applications*, IGI Global, 2017, pp. 324–350.
RAN 14	R. Rana, M. Staron, J. Hansson, M. Nilsson, and W. Meding, "A framework for adoption of machine learning in industry for software defect prediction," in ICSOFT-EA 2014 - Proceedings of the 9th International Conference on Software Engineering and Applications, 2014, pp. 383–392.
RAT 15	S. S. Rathore and S. Kumar, "Predicting number of faults in software system using genetic programming," *Procedia Comput. Sci.*, vol. 62, pp. 303–311, 2015.
RAT 16	S. S. Rathore and S. Kumar, "A Decision Tree Regression Based Approach for the Number of Software Faults Prediction," *SIGSOFT Softw. Eng. Notes*, vol. 41, no. 1, pp. 1–6, 2016.
RAT 17	S. S. Rathore and S. Kumar, "A study on software fault prediction techniques," *Artif. Intell. Rev.*, vol. 51, no. 2, pp. 255–327, 2017.
RAT 17 A	S. S. Rathore and S. Kumar, "An empirical study of some software fault prediction techniques for the number of faults prediction," *Soft Comput.*, vol. 21, no. 24, pp. 7417–7434, 2017.
RAT 21	S. S. Rathore and S. Kumar, "An empirical study of ensemble techniques for software fault prediction," *Appl. Intell.*, vol. 51, no. 6, pp. 3615–3644, 2021.
RHM 20	W. Rhmann, B. Pandey, G. Ansari, and D. K. Pandey, "Software fault prediction based on change metrics using hybrid algorithms: An empirical study," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 32, no. 4, pp. 419–424, 2020.
RIT 21	O. P. Ritu, "Software Quality Prediction Method Using Fuzzy Logic," *Turkish J. Comput. Math. Educ. Vol*, vol. 12, no. 11, pp. 807–817, 2021.
ROS 61	F. Rosenblatt, "Principles of neurodynamics. perceptrons and the theory of brain mechanisms," 1961.

ROY 70	D. W. W. Royce, “Managing the Development of large Software Systems,” in *Ieee Wescon*, 1970, no. August, pp. 1–9.
RYU 15	D. Ryu, J. I. Jang, and J. Baik, “A Hybrid Instance Selection Using Nearest-Neighbor for Cross-Project Defect Prediction,” *J. Comput. Sci. Technol.*, vol. 30, no. 5, pp. 969–980, 2015.
SAI 11	M. Saidi, “Traitement de données médicales par un Système Immunitaire Artificiel Reconnaissance Automatique du Diabète,” *Mémoire Magister, Univ. Aboubakr Belkaid, Tlemcen, Algérie*, 2011.
SAM 10	D. Samadhiya, S. H. Wang, and D. Chen, “Quality models: Role and value in software engineering,” in *ICSTE 2010 - 2010 2nd International Conference on Software Technology and Engineering, Proceedings*, 2010, vol. 1, pp. V1--320.
SAR 12	F. Sarro, S. Di Martino, F. Ferrucci, and C. Gravino, “A further analysis on the use of genetic algorithm to configure support vector machines for inter-release fault prediction,” in Proceedings of the ACM Symposium on Applied Computing, 2012, pp. 1215–1220.
SAY 05	S. Sayyad and T. Menzies, “The PROMISE Repository of Software Engineering Databases, University of Ottowa, Kanada,” *Sch. Inf. Technol. Eng. Univ. Ottawa, Canada*, vol. 24, 2015. http://promise.site.uottawa.ca/SERepository/datasets-page.html
SCH 06	A. Schröter, T. Zimmermann, R. Premraj, and A. Zeller, “If your bug database could talk,” in *Proceedings of the 5th international symposium on empirical software engineering*, 2006, vol. 2, pp. 18–20.
SCH 07	G. G. Schulmeyer and J. I. McManus, *Handbook of software quality assurance (3rd ed.)*. Artech House, Inc., 2007.
SCH 92	N. F. Schneidewind, “Methodology for Validating Software Metrics,” *IEEE Trans. Softw. Eng.*, vol. 18, no. 5, pp. 410–422, 1992.
SHA 08	R. Shatnawi and W. Li, “The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process,” *J. Syst. Softw.*, vol. 81, no. 11, pp. 1868–1882, 2008.
SHA 11	P. M. Shanthi and K. Duraiswamy, “An empirical validation of software quality metric suites on open source software for fault-proneness prediction in object oriented systems,” *Eur. J. Sci. Res.*, vol. 51, no. 2, pp. 168–181, 2011.
SHA 14	S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.
SHA 19	D. Sharma and P. Chandra, “A comparative analysis of soft computing techniques in software fault prediction model development,” *Int. J. Inf. Technol.*, vol. 11, no. 1, pp. 37–46, 2019.
SHA 20	D. Sharma and P. Chandra, “Linear regression with factor analysis in fault prediction of software,” *J. Interdiscip. Math.*, vol. 23, no. 1, pp. 11–19, 2020.
SHA 21	S. Sharma and S. Vijayvargiya, “Applying Soft Computing Techniques for Software Project Effort Estimation Modelling,” in *Nanoelectronics, Circuits and Communication Systems*, Springer, 2021, pp. 211–227.
SHE 14	M. Shepperd, D. Bowes, and T. Hall, “Researcher bias: The use of machine learning in software defect prediction,” *IEEE Trans. Softw. Eng.*, vol. 40, no. 6, pp. 603–616, 2014.
SHE 95	S. A. Sherer, “Software fault prediction,” *J. Syst. Softw.*, vol. 29, no. 2, pp. 97–105, 1995.
SIN 09	Y. Singh, A. Kaur, and R. Malhotra, “Software Fault Proneness Prediction Using Support Vector Machines,” in *Lecture Notes in Engineering and Computer Science*, 2009, vol. 2176, no. 1, pp. 240–245.

SIN 10	Y. Singh, A. Kaur, and R. Malhotra, "Empirical validation of object-oriented metrics for predicting fault proneness models," *Softw. Qual. J.*, vol. 18, no. 1, pp. 3–35, 2010.
SPI 05	D. Spinellis, "Tool writing: A forgotten art?," *IEEE Softw.*, vol. 22, no. 4, pp. 9–11, 2005.
TAN 16	P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to data mining*. Pearson Education India, 2016.
TAN 99	M.-H. Tang, M.-H. Kao, and M.-H. Chen, "An empirical study on object-oriented metrics," in *Software Metrics Symposium, 1999. Proceedings. Sixth International*, 1999, pp. 242–249.
TIA 05	J. Tian, *Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement*. John Wiley \& Sons, 2005.
TIM 00	J. Timmis, M. Neal, and J. Hunt, "An artificial immune system for data analysis," *BioSystems*, vol. 55, no. 1–3, pp. 143–150, 2000.
TIM 01	J. Timmis and M. Neal, "A resource limited artificial immune system for data analysis," in *Knowledge-Based Systems*, vol. 14, no. 3–4, Springer, 2001, pp. 121–130.
TIM 02	J. Timmis and T. Knight, "Artificial immune systems: Using the immune system as inspiration for data mining," in *Data mining: A heuristic approach*, IGI Global, 2002, p. 209.
TRI 17	I. Triguero, S. González, J. M. Moyano, S. García, J. Alcalá-Fdez, J. Luengo, A. Fernández, M. J. del Jesus, L. Sánchez, F. Herrera.., "KEEL 3.0: An Open Source Software for Multi-Stage Analysis in Data Mining," Int. J. Comput. Intell. Syst., vol. 10, no. 1, p. 1238, 2017.
TUR 09	B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empir. Softw. Eng.*, vol. 14, no. 5, pp. 540–578, Oct. 2009.
TUR 11	B. Turhan, A. Tosun, and A. Bener, "Empirical evaluation of mixed-project defect prediction models," in *Proceedings - 37th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2011*, 2011, pp. 396–403.
TUR 36	A. M. Turing, "On computable numbers, with an application to the entscheidungsproblem," *Proc. London Math. Soc.*, vol. s2-42, no. 1, pp. 230–265, 1937.
TUR 50	A. M. Turing, "Computing machinery and intelligence," *Mind , Vol. 49*, pp. 433–460, 1950.
VAN 08	O. Vandecruys, D. Martens, B. Baesens, C. Mues, M. De Backer, and R. Haesen, "Mining software repositories for comprehensible software fault prediction models," *J. Syst. Softw.*, vol. 81, no. 5, pp. 823–839, 2008.
VER 89	F. T. Vertosick and R. H. Kelly, "Immune network theory: A role for parallel distributed processing?," *Immunology*, vol. 66, no. 1, pp. 1–7, 1989.
WAH	R. S. Wahono and N. S. Herman, "Genetic feature selection for software defect prediction," *Adv. Sci. Lett.*, vol. 20, no. 1, pp. 239–244, 2014.
WAN 03	J. Wang, *Data mining: opportunities and challenges*. Idea Group Pub., 2003.
WAT 01	A. Watkins, "AIRS: A Resource Limited Artificial Immune Classifier, Master Thesis," Mississippi State University,USA, 2001.
WAT 02	A. Watkins and J. Timmis, "Artificial Immune Recognition System (AIRS): Revisions and Refinements," in *1st International Conference on Artificial Immune Systems*, 2002, pp. 173–181.
WAT 03	A. Watkins, X. Bi, and A. Phadke, "Parallelizing an immune-inspired algorithm for efficient pattern recognition," Intell. Eng. Syst. through Artif. Neural Networks Smart Eng. Syst. Des. Neural Networks, Fuzzy Logic, Evol. Program. Complex Syst. Artif. Life, vol. 13, pp. 225–230, 2003.

WAT 04	A. Watkins and J. Timmis, “Exploiting parallelism inherent in AIRS an artificial immune classifier,” in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2004, vol. 3239, pp. 427–438.
WEY 10	E. J. Weyuker, T. J. Ostrand, and R. M. Bell, “Comparing the effectiveness of several modeling methods for fault prediction,” *Empir. Softw. Eng.*, vol. 15, no. 3, pp. 277–295, 2010.
WHI 03	J. A. White and S. M. Garrett, “Improved pattern recognition with artificial clonal selection?,” in *International Conference on Artificial Immune Systems*, 2003, pp. 181–193.
WIR 08	N. Wirth, “A Brief History of Software Engineering,” *IEEE Ann. Hist. Comput.*, vol. 30, no. 3, pp. 32–39, 2008.
WIT 16	I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, “Data Mining: Practical Machine Learning Tools and Techniques,” *Data Min. Pract. Mach. Learn. Tools Tech.*, pp. 1–621, 2016.
XU 07	Z. Xu, X. Zheng, and P. Guo, “Empirically validating software metrics for risk prediction based on intelligent methods,” in *Journal of Digital Information Management*, 2007, vol. 5, no. 3, pp. 99–106.
YOH 17	C. W. Yohannese, T. Li, M. Simfukwe, and F. Khurshid, “Ensembles based combined learning for improved software fault prediction: A comparative study,” in Proceedings of the 2017 12th International Conference on Intelligent Systems and Knowledge Engineering, ISKE 2017, 2017, pp. 1–6.
ZEK 15	M. ZEKRI, “Approches Bio-inspirées pour la Fouille de Données en Bioinformatique,” Thése de Doctorat, Université Badji Mokhtar, Annaba, Algérie, 2015.
ZHA 03	D. Zhang and J. J. P. Tsai, “Machine learning and software engineering,” *Softw. Qual. J.*, vol. 11, no. 2, pp. 87–119, 2003.
ZHA 09	H. Zhang, “An investigation of the relationships between lines of code and defects,” in *2009 IEEE International Conference on Software Maintenance*, 2009, pp. 274–283.
ZHO 04	S. Zhong, T. M. Khoshgoftaar, and N. Seliya, “Unsupervised learning for expert-based software quality estimation,” in *Proceedings of IEEE International Symposium on High Assurance Systems Engineering*, 2004, vol. 8, pp. 149–155.
ZHO 08	Y. Zhou and H. Leung, “Empirical analysis of object-oriented design metrics for predicting high and low severity faults,” *IEEE Trans. Softw. Eng.*, vol. 32, no. 10, pp. 771–789, 2006.
ZHO 10	Y. Zhou, B. Xu, and H. Leung, “On the ability of complexity metrics to predict fault-prone classes in object-oriented systems,” *J. Syst. Softw.*, vol. 83, no. 4, pp. 660–674, 2010.
ZIG 18	N. Zighed, N. Bounour, and A.-D. Seriai, “Comparative analysis of object-oriented software maintainability prediction models,” *Found. Comput. Decis. Sci.*, vol. 43, no. 4, pp. 359–374, 2018.
ZIM 09	T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, “Cross-project defect prediction: A large scale experiment on data vs. domain vs. process,” in ESEC-FSE’09 - Proceedings of the Joint 12th European Software Engineering Conference and 17th ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2009, pp. 91–100.
1	www.images.math.cnrs.frNos-neurones-se-synchronisent-ils.html
2	http://keepschool.com/fiches-de-cours/lycee/svt-biologie/systeme-immunitaire.html
3	https://www.thepartnershipineducation.com/resources/immune-system
4	https://www.slideshare.net/lndecastro/2000-artificial-immune-systems-theory-and-applications

5	https://www.cs.waikato.ac.nz/ml/weka/
6	https://www.mathworks.com/products/matlab.html
7	http://www.keel.es/
8	https://www.tensorflow.org/

Annexe

Les tables A1, A2 et A3 représente les résultats détaillés de l'expérimentation qui a pour but d'évaluer et comparer les systèmes immunitaires artificiels pour construire des modèles de prédiction de défauts logiciels Inter-projets.

Table A. 1- Résultats détaillés de l'expérimentation en Inter-Projets en terme de Précision

Learning DATA	ant 1.3	ant 1.4	ant 1.5	ant 1.6	camel 1.0	camel 1.2	camel 1.4	ivy 1.1	ivy 1.4
Test DATA	ant 1.4	ant 1.5	ant 1.6	ant 1.7	camel 1.2	camel 1.4	camel 1.6	ivy 1.4	ivy 2.0
Algorithm	Précision	Précision	Précision	Précision	Précision	Précision	Précision	Précision	Précision
AIRS1	0.273	0.194	0.484	0.356	0.5	0.179	0.331	0.132	0
AIRS2	0.194	0.25	0.704	0.514	0.7	0.192	0.278	0.087	0.267
AIRS2 parallel	0.333	0.132	0.591	0.528	0.25	0.268	0.463	0.112	0.25
CLONALG	0.294	0	0	0.1	0	0.207	0	0.113	0
CSCA	0	0.286	0.556	0.631	0	0.284	0.288	0.104	0.2
Immunos 1	0.23	0.123	0.299	0.26	0.455	0.21	0.205	0.074	0.171
Immunos 2	0	0	0	0.457	0	0	0	0.066	0
Immunos 99	0.236	0.114	0.291	0.252	0	0.286	0.222	0.073	0.171
J48	0.28	0.185	0.7	0.62	0	0.381	0.389	0.104	0
NB	0.316	0.128	0.5	0.539	0.628	0.333	0.379	0.169	0.432
MLP	0.308	0.211	0.727	0.438	0.667	0.29	0.494	0.094	0.25
RF	0.333	0.159	0.667	0.496	0.7	0.383	0.51	0.126	0
SVM	0.196	0.247	0.548	0.171	0.556	0.208	0.299	0.075	0.192
LR	0.318	0.278	0.667	0.614	0.75	0.352	0.489	0.095	0.2
IBK	0.296	0.139	0.583	0.451	0.476	0.333	0.466	0.107	0.071
Learning DATA	jedit 3.2	jedit 4.0	jedit 4.1	jedit 4.2	log4j 1.0	log4j 1.1	lucene 2.0	lucene 2.2	poi 1.5
Test DATA	jedit 4.0	jedit 4.1	jedit 4.2	jedit 4.3	log4j 1.1	log4j 1.2	lucene 2.2	lucene 2.4	poi 2.0
Algorithm	Précision	Précision	Précision	Précision	Précision	Précision	Précision	Précision	Précision
AIRS1	0.391	0.468	0.391	0.05	0.75	0.957	0.664	0.603	0.112
AIRS2	0.459	0.673	0.371	0.088	0.85	0.934	0.714	0.647	0.099
AIRS2 parallel	0.469	0.638	0.475	0.108	0.8	0.967	0.67	0.685	0.075
CLONALG	0.443	0.644	0.308	0.2	0.7	0.95	0.567	0.67	0.139
CSCA	0.458	0.646	0.372	0.108	0.647	0.933	0.689	0.673	0.117
Immunos 1	0.276	0.307	0.179	0.03	0.362	0.946	0.611	0.626	0.124
Immunos 2	0.5	0	0.4	0	0	1	0.631	0.597	0.119
Immunos 99	0.268	0.311	0.176	0.052	0.391	0.915	0.599	0.632	0.124
J48	0.527	0.58	0.418	0.1	0.773	0.978	0.721	0.622	0.147
NB	0.513	0.648	0.419	0.082	0.821	0.966	0.779	0.798	0.216
MLP	0.436	0.667	0.376	0.135	0.792	0.926	0.681	0.66	0.108
RF	0.559	0.627	0.449	0.132	0.767	0.949	0.729	0.641	0.133
SVM	0.417	0.218	0.099	0.025	0.613	0.93	0.673	0.586	0.124
LR	0.5	0.745	0.472	0.143	0.786	0.968	0.742	0.655	0.1
IBK	0.51	0.632	0.352	0.077	0.742	0.955	0.733	0.676	0.134

Learning DATA	poi 2.0	poi 2.5	synapse 1.0	synapse 1.1	velocity 1.4	velocity 1.5	xalan 2.4	xalan 2.5	xalan 2.6
Test DATA	poi 2.5	poi 3.0	synapse 1.1	synapse 1.2	velocity 1.5	velocity 1.6	xalan 2.5	xalan 2.6	xalan 2.7
Algorithm	Précision	Précision	Précision	Précision	Précision	Précision	Précision	Précision	Précision
AIRS1	0.409	0.743	0.385	0.475	0.66	0.399	0.653	0.603	1
AIRS2	0.778	0.726	0.267	0.423	0.668	0.377	0.554	0.537	1
AIRS2 parallel	1	0.722	0.571	0.583	0.654	0.374	0.577	0.496	0.997
CLONALG	0.245	0.636	0	0.339	0.667	0.455	0.698	0.66	1
CSCA	0.813	0.761	0	0.659	0.632	0.398	0.765	0.612	1
Immunos 1	0.574	0.663	0.288	0.366	0.554	0.407	0.537	0.501	0.988
Immunos 2	0	0.636	0	0	0.664	0.341	0	0.568	0.994
Immunos 99	0.632	0.667	0.29	0.361	0.591	0.369	0.534	0.485	0.988
J48	0.688	0.709	0.529	0.565	0.658	0.413	0.632	0.541	1
NB	0.774	0.88	0.591	0.6	0.653	0.567	0.608	0.706	1
MLP	0.717	0.728	0.5	0.578	0.66	0.397	0.675	0.624	1
RF	0.697	0.719	0.462	0.628	0.665	0.445	0.671	0.594	1
SVM	0.712	0.621	0.359	0.515	0.646	0.394	0.466	0.399	1
LR	0.889	0.708	0.526	0.592	0.658	0.409	0.7	0.537	1
IBK	0.69	0.689	0.438	0.556	0.656	0.432	0.64	0.627	1

Learning DATA	xerces-init	xerces 1.2	xerces 1.3
Test DATA	xerces 1.2	xerces 1.3	xerces 1.4
Algorithm	Précision	Précision	Précision
AIRS1	0.14	0.075	0.926
AIRS2	0.195	0.194	0.941
AIRS2 parallel	0.181	0.125	0.97
CLONALG	0.184	0.13	1
CSCA	0.2	0.167	0.939
Immunos 1	0.168	0.117	0.89
Immunos 2	0.129	0	0
Immunos 99	0.161	0.01	0.89
J48	0.24	0.269	0.989
NB	0.227	0.354	0.963
MLP	0.221	0.241	0.984
RF	0.271	0.182	0.944
SVM	0.193	0.326	0.894
LR	0.185	0.727	1
IBK	0.218	0.214	0.951

Table A. 2- Résultats détaillés de l'expérimentation en Inter-Projets en terme de Rappel

Learning DATA	ant 1.3	ant 1.4	ant 1.5	ant 1.6	camel 1.0	camel 1.2	camel 1.4	ivy 1.1
Test DATA	ant 1.4	ant 1.5	ant 1.6	ant 1.7	camel 1.2	camel 1.4	camel 1.6	ivy 1.4
Algorithmes	Rappel	Rappel	Rappel	Rappel	Rappel	Rappel	Rappel	Rappel
AIRS1	0.15	0.375	0.163	0.47	0.042	0.49	0.223	0.75
AIRS2	0.175	0.344	0.207	0.536	0.032	0.407	0.186	0.5
AIRS2 parallel	0.1	0.219	0.283	0.404	0.032	0.228	0.101	0.75
CLONALG	0.125	0	0	0.036	0	0.517	0	0.813
CSCA	0	0.188	0.109	0.536	0	0.448	0.09	0.875
Immunos 1	0.85	1	0.989	0.988	0.255	0.772	0.739	0.875
Immunos 2	0	0	0	0.127	0	0	0	0.938
Immunos 99	0.95	1	0.989	0.988	0	0.49	0.356	0.875
J48	0.175	0.313	0.304	0.452	0	0.407	0.223	0.813
NB	0.3	0.719	0.641	0.5	0.125	0.331	0.25	0.625
MLP	0.2	0.375	0.174	0.596	0.028	0.586	0.202	0.75
RF	0.15	0.219	0.217	0.422	0.032	0.6	0.261	0.875
SVM	0.55	0.656	0.435	0.657	0.069	0.517	0.261	0.438
LR	0.175	0.313	0.217	0.47	0.028	0.303	0.117	0.5
IBK	0.2	0.313	0.228	0.476	0.046	0.628	0.404	0.75
Learning DATA	ivy 1.4	jedit 3.2	jedit 4.0	jedit 4.1	jedit 4.2	log4j 1.0	log4j 1.1	lucene 2.0
Test DATA	ivy 2.0	jedit 4.0	jedit 4.1	jedit 4.2	jedit 4.3	log4j 1.1	log4j 1.2	lucene 2.2
Algorithmes	Rappel	Rappel	Rappel	Rappel	Rappel	Rappel	Rappel	Rappel
AIRS1	0	0.6	0.468	0.563	0.455	0.568	0.238	0.618
AIRS2	0.1	0.667	0.443	0.542	0.455	0.459	0.376	0.521
AIRS2 parallel	0.025	0.507	0.468	0.604	0.364	0.432	0.307	0.507
CLONALG	0	0.627	0.367	0.333	0.182	0.568	0.201	0.646
CSCA	0.025	0.36	0.532	0.604	0.364	0.595	0.222	0.507
Immunos 1	0.925	0.947	0.924	0.938	0.636	0.919	0.741	0.861
Immunos 2	0	0.253	0	0.042	0	0	0.026	0.653
Immunos 99	0.925	0.947	0.937	0.938	0.364	0.919	0.91	0.882
J48	0	0.64	0.506	0.583	0.455	0.459	0.238	0.431
NB	0.4	0.52	0.443	0.542	0.364	0.622	0.296	0.368
MLP	0.075	0.813	0.456	0.667	0.455	0.514	0.333	0.535
RF	0	0.693	0.468	0.646	0.455	0.622	0.296	0.486
SVM	0.125	0.533	0.646	0.688	0.909	0.514	0.28	0.528
LR	0.025	0.627	0.443	0.708	0.455	0.595	0.317	0.5
IBK	0.025	0.693	0.544	0.646	0.364	0.622	0.339	0.535
Learning DATA	lucene 2.2	poi 1.5	poi 2.0	poi 2.5	synapse 1.0	synapse 1.1	velocity 1.4	velocity 1.5
Test DATA	lucene 2.4	poi 2.0	poi 2.5	poi 3.0	synapse 1.1	synapse 1.2	velocity 1.5	velocity 1.6
Algorithmes	Rappel	Rappel	Rappel	Rappel	Rappel	Rappel	Rappel	Rappel
AIRS1	0.621	0.676	0.036	0.712	0.167	0.337	0.93	0.808

AIRS2	0.443	0.486	0.056	0.772	0.133	0.384	0.965	0.705
AIRS2 parallel	0.493	0.324	0.036	0.73	0.267	0.407	0.944	0.744
CLONALG	0.759	0.784	0.052	1	0	0.233	0.718	0.833
CSCA	0.66	0.622	0.052	0.769	0	0.337	0.859	0.872
Immunos 1	0.916	0.973	0.456	0.975	0.883	0.965	0.218	0.731
Immunos 2	1	1	0	1	0	0	1	1
Immunos 99	0.921	0.946	0.048	0.975	0.933	0.965	0.183	0.846
J48	0.892	0.784	0.089	0.598	0.15	0.302	0.937	0.795
NB	0.409	0.432	0.097	0.313	0.433	0.488	0.887	0.436
MLP	0.69	0.595	0.153	0.619	0.2	0.302	0.93	0.795
RF	0.626	0.703	0.093	0.502	0.1	0.314	0.937	0.833
SVM	0.803	0.676	0.21	0.911	0.233	0.395	0.887	0.667
LR	0.739	0.568	0.032	0.786	0.167	0.337	0.923	0.859
IBK	0.596	0.676	0.117	0.584	0.233	0.291	0.901	0.821
Learning DATA	xalan 2.4	xalan 2.5	xalan 2.6	xerces-init	xerces 1.2	xerces 1.3		
Test DATA	xalan 2.5	xalan 2.6	xalan 2.7	xerces 1.2	xerces 1.3	xerces 1.4		
Algorithmes	Rappel	Rappel	Rappel	Rappel	Rappel	Rappel		
AIRS1	0.121	0.533	0.37	0.352	0.13	0.114		
AIRS2	0.08	0.499	0.45	0.577	0.174	0.183		
AIRS2 parallel	0.078	0.479	0.317	0.549	0.058	0.146		
CLONALG	0.096	0.764	0.421	0.563	0.435	0.034		
CSCA	0.101	0.659	0.412	0.62	0.029	0.105		
Immunos 1	0.811	0.961	0.824	0.437	0.174	0.705		
Immunos 2	0	0.783	0.559	0.38	0	0		
Immunos 99	0.811	0.927	0.833	0.437	0.014	0.703		
J48	0.111	0.611	0.478	0.732	0.203	0.211		
NB	0.204	0.304	0.273	0.211	0.246	0.178		
MLP	0.14	0.732	0.374	0.789	0.101	0.142		
RF	0.121	0.623	0.458	0.831	0.116	0.117		
SVM	0.744	0.655	0.237	0.803	0.217	0.096		
LR	0.127	0.438	0.401	0.789	0.116	0.117		
IBK	0.183	0.703	0.4	0.648	0.174	0.178		

Table A. 3- Résultats détaillés de l'expérimentation en Inter-Projets en terme de f-mesure

Learning DATA	ant 1.3	ant 1.4	ant 1.5	ant 1.6	camel 1.0	camel 1.2	camel 1.4	ivy 1.1
Test DATA	ant 1.4	ant 1.5	ant 1.6	ant 1.7	camel 1.2	camel 1.4	camel 1.6	ivy 1.4
Algorithmes	*F1*	*F1*	*F1*	*F1*	*F1*	*F1*	*F1*	*F1*
AIRS1	0.194	0.256	0.244	0.405	0.077	0.262	0.266	0.224
AIRS2	0.184	0.290	0.320	0.525	0.061	0.261	0.223	0.148
AIRS2 parallel	0.154	0.165	0.383	0.458	0.057	0.246	0.166	0.195
CLONALG	0.175	0.000	0.000	0.053	0.000	0.296	0.000	0.198

CSCA	0.000	0.227	0.182	0.580	0.000	0.348	0.137	0.186
Immunos 1	0.362	0.219	0.459	0.412	0.327	0.330	0.321	0.136
Immunos 2	0.000	0.000	0.000	0.199	0.000	0.000	0.000	0.123
Immunos 99	0.378	0.205	0.450	0.402	0.000	0.361	0.273	0.135
J48	0.215	0.233	0.424	0.523	0.000	0.394	0.283	0.184
NB	0.308	0.217	0.562	0.519	0.208	0.332	0.301	0.266
MLP	0.243	0.270	0.281	0.505	0.054	0.388	0.287	0.167
RF	0.207	0.184	0.327	0.456	0.061	0.468	0.345	0.220
SVM	0.289	0.359	0.485	0.271	0.123	0.297	0.279	0.128
LR	0.226	0.294	0.327	0.532	0.054	0.326	0.189	0.160
IBK	0.239	0.193	0.328	0.463	0.084	0.435	0.433	0.187
Learning DATA	ivy 1.4	jedit 3.2	jedit 4.0	jedit 4.1	jedit 4.2	log4j 1.0	log4j 1.1	lucene 2.0
Test DATA	ivy 2.0	jedit 4.0	jedit 4.1	jedit 4.2	jedit 4.3	log4j 1.1	log4j 1.2	lucene 2.2
Algorithmes	*F1*	*F1*	*F1*	*F1*	*F1*	*F1*	*F1*	*F1*
AIRS1	0.000	0.473	0.468	0.461	0.090	0.646	0.381	0.640
AIRS2	0.146	0.544	0.534	0.440	0.147	0.596	0.536	0.602
AIRS2 parallel	0.045	0.487	0.540	0.532	0.167	0.561	0.466	0.577
CLONALG	0.000	0.519	0.468	0.320	0.191	0.627	0.332	0.604
CSCA	0.044	0.403	0.583	0.460	0.167	0.620	0.359	0.584
Immunos 1	0.289	0.427	0.461	0.301	0.057	0.519	0.831	0.715
Immunos 2	0.000	0.336	0.000	0.076	0.000	0.000	0.051	0.642
Immunos 99	0.289	0.418	0.467	0.296	0.091	0.549	0.912	0.713
J48	0.000	0.578	0.540	0.487	0.164	0.576	0.383	0.539
NB	0.415	0.516	0.526	0.473	0.134	0.708	0.453	0.500
MLP	0.115	0.568	0.542	0.481	0.208	0.623	0.490	0.599
RF	0.000	0.619	0.536	0.530	0.205	0.687	0.451	0.583
SVM	0.151	0.468	0.326	0.173	0.049	0.559	0.430	0.592
LR	0.044	0.556	0.556	0.566	0.218	0.677	0.478	0.597
IBK	0.037	0.588	0.585	0.456	0.127	0.677	0.500	0.619
Learning DATA	lucene 2.2	poi 1.5	poi 2.0	poi 2.5	synapse 1.0	synapse 1.1	velocity 1.4	velocity 1.5
Test DATA	lucene 2.4	poi 2.0	poi 2.5	poi 3.0	synapse 1.1	synapse 1.2	velocity 1.5	velocity 1.6
Algorithmes	*F1*	*F1*	*F1*	*F1*	*F1*	*F1*	*F1*	*F1*
AIRS1	0.612	0.192	0.066	0.727	0.233	0.394	0.772	0.534
AIRS2	0.526	0.164	0.104	0.748	0.178	0.403	0.789	0.491
AIRS2 parallel	0.573	0.122	0.069	0.726	0.364	0.479	0.773	0.498
CLONALG	0.712	0.236	0.086	0.778	0.000	0.276	0.692	0.589
CSCA	0.666	0.197	0.098	0.765	0.000	0.446	0.728	0.547
Immunos 1	0.744	0.220	0.508	0.789	0.434	0.531	0.313	0.523
Immunos 2	0.748	0.213	0.000	0.778	0.000	0.000	0.798	0.509
Immunos 99	0.750	0.219	0.089	0.792	0.442	0.525	0.279	0.514

J48	0.733	0.248	0.158	0.649	0.234	0.394	0.773	0.544
NB	0.541	0.288	0.172	0.462	0.500	0.538	0.752	0.493
MLP	0.675	0.183	0.252	0.669	0.286	0.397	0.772	0.530
RF	0.633	0.224	0.164	0.591	0.164	0.419	0.778	0.580
SVM	0.678	0.210	0.324	0.739	0.283	0.447	0.748	0.495
LR	0.694	0.170	0.062	0.745	0.254	0.430	0.768	0.554
IBK	0.633	0.224	0.200	0.632	0.304	0.382	0.759	0.566

Learning DATA	xalan 2.4	xalan 2.5	xalan 2.6	xerces-init	xerces 1.2	xerces 1.3
Test DATA	xalan 2.5	xalan 2.6	xalan 2.7	xerces 1.2	xerces 1.3	xerces 1.4
Algorithmes	*F1*	*F1*	*F1*	*F1*	*F1*	*F1*
AIRS1	0.204	0.566	0.540	0.200	0.095	0.203
AIRS2	0.140	0.517	0.621	0.291	0.183	0.306
AIRS2 parallel	0.137	0.487	0.481	0.272	0.079	0.254
CLONALG	0.169	0.708	0.593	0.277	0.200	0.066
CSCA	0.178	0.635	0.584	0.302	0.049	0.189
Immunos 1	0.646	0.659	0.899	0.243	0.140	0.787
Immunos 2	0.000	0.658	0.716	0.193	0.000	0.000
Immunos 99	0.644	0.637	0.904	0.235	0.012	0.786
J48	0.189	0.574	0.647	0.361	0.231	0.348
NB	0.305	0.425	0.429	0.219	0.290	0.300
MLP	0.232	0.674	0.544	0.345	0.142	0.248
RF	0.205	0.608	0.628	0.409	0.142	0.208
SVM	0.573	0.496	0.383	0.311	0.261	0.173
LR	0.215	0.482	0.572	0.300	0.200	0.209
IBK	0.285	0.663	0.571	0.326	0.192	0.300

A PROPOS DE L'AUTEUR

BIOGRAPHIE DE L'AUTEUR

Ahmed Taha HAOUARI né le 4 juillet 1992 à Annaba, où il a poursuivi ses études jusqu'à l'obtention de son baccalauréat scientifique en 2011. L'année d'après, il se dirige vers une formation universitaire au département de Mathématiques et Informatique de l'université Badji Mokhtar d'Annaba. Il décide, par la suite, de continuer ses études universitaires au département d'informatique où il a obtenu sa licence sous l'encadrement de Dr Benouhiba Toufik en 2014. Après mûre réflexion, il prend la décision de poursuivre sa formation académique et rejoint la formation de Master en Informatique dans la spécialité Ingénierie des Logiciels Complexes (ILC). Il sera diplômé en 2016 sous l'encadrement de Pr Souici-Meslati Labiba.

En octobre 2016, il passe son concours pour accéder à une formation doctorale spécialisée dans l'Ingénierie des Systèmes Complexes (ISC), affiliée au laboratoire LISCO et sera supervisé par Pr Atil Fadila et Pr Souici-Meslati Labiba. Le thème choisi s'intéresse à l'utilisation des méthodes bio-inspirées de fouilles de données pour la résolution de problèmes liés au génie logiciels. Parallèlement à sa formation doctorale, l'auteur fait ses débuts en qualité d'enseignant, d'abord au département de biologie où il a enseigné le langage R, puis au département de Sciences et Techniques pour apprendre à ses étudiants les rudiments du langage C et enfin au département Mathématiques et Informatique où il sera responsable des Travaux pratique (TP) en algorithmique et structure de données par le langage de programmation C. Pour enrichir son expérience, il s'essaie à l'enseignement privé en intégrant, pour quelque mois, l'Institut Spécialisé des Métiers (ISM) se spécialisant dans l'introduction des concepts des réseaux informatiques.

Dans le cadre de ses activités scientifiques dans le domaine de la prédiction de défauts logiciels, Mr Haouari a participé à trois conférences internationales. Il a aussi publié un article dans un journal international de renommée établie. Il est, depuis 2016, membre du Laboratoire d'Ingénierie des Systèmes Complexes (LISCO) de l'université d'Annaba.

Publication Internationale

Haouari A. T., Souici-Meslati L., Atil F, Meslati D., "Empirical comparison and evaluation of Artificial Immune Systems in inter-release software fault prediction", Applied Soft Computing Journal, Vol.96, 106686, Sept 2020,

https://www.sciencedirect.com/science/article/abs/pii/S1568494620306244, https://doi.org/10.1016/j.asoc.2020.106686

Informations sur Applied Soft Computing Journal

ISSN: 1568-4946, **EISSN** : 1872-9681

Publié par: ELSEVIER, depuis 2001

Site Web: https://www.journals.elsevier.com/applied-soft-computing

Indexé par : SCOPUS, WEB OF SCIENCE Current Contents - Engineering, Computing & Technology, Science Citation Index Expanded (Clarivate Analytic), Journal Citation Report (JCR).

H Index: 143, **Quartile**: Q1 (http://www.scimagojr.com/)

Impact Factor: 6,725 (Journal Citation Report JCR, Web of Science)

Ranking : 11 out of 112 in Computer Science, Interdisciplinary Applications

Communications internationales (3)

1 A. T. Haouari, L. Souici-Meslati, F. Atil, "Outil d'aide à la Prédiction des Défauts Logiciels", ASD'2017, 11ème Conférence sur les Avancées des Systèmes Décisionnels, Tabarka, Tunisie, 2017

2 A. T. Haouari, L. Souici-Meslati, F. Atil, "Systèmes Immunitaires Artificiels pour la Prédiction de Défauts Logiciels," COSI'2017, 14ème Colloque sur l'Optimisation et les Systèmes d'Information, Bouira, Algérie, 2017.

3 Haouari A. T., Souici-Meslati L., Atil F., "Prédiction de défauts logiciels par des classifieurs immunologiques", 14èmes rencontres de la Société Francophone de Classification, SFC'2017, pp. 73-76, Lyon, France, 2017