Université de Montréal

# Estimation de caractéristiques externes de qualité à partir de mesures d'attributs internes. Bilan et perspectives

Par

Lynda Ait Mehedine

Département d'informatique et
de recherche opérationnelle

Faculté des arts et sciences

Mémoire présenté à la Faculté des études supérieures
En vue de l'obtention du grade de
**Maître ès sciences(M. Sc.)**
en informatique

Janvier 2005



Université de Montréal

Faculté des études supérieures

Ce mémoire intitulé :

Estimation de caractéristiques externes de qualité à partir de mesures d'attributs internes.

Bilan et perspectives

Présenté par :

Lynda Ait Mehedine

a été évalué par un jury composé des personnes suivantes :

Stefan Monnier
(président-rapporteur)

Houari sahraoui
(directeur de maîtrise)

Hakim Lounis
(codirecteur de maîtrise)

Sylvie Hamel
(membre du jury)

Mémoire accepté le 08/03/05

# Sommaire

De nos jours, nous utilisons la technologie Orientée Objet (OO). Elle offre une qualité du logiciel supérieure à ce qu'elle était dans le paradigme procédural. Elle permet aussi de réduire les coûts de développement du logiciel et de mettre au point des logiciels de qualité.

Sachant que la qualité est le critère majeur pour l'acceptation et le succès d'un logiciel, on essaye dans ce travail de trouver s'il existe un lien entre des mesures internes faites sur des composants orientés objets et trois facteurs de qualité, qui sont la propension d'avoir des erreurs, le coût de la maintenance corrective et la réutilisabilité.

Dans cette recherche, nous avons pu obtenir et exploiter des mesures recueillies tout au long du cycle de développement de quelques logiciels. Différents algorithmes d'apprentissage sont explorés à travers leurs capacités de produire des modèles prédictifs. La qualité de chaque modèle est alors évaluée et chaque mesure interne pertinente pour la prédiction d'un critère de qualité sera identifiée.

Mots clés : qualité de logiciel, techniques d'estimation, apprentissage automatique.

# Abstract

Nowadays, we use the Object Oriented (OO) technology. It allows faster development and higher quality of the produces software than the procedural paradigm. It also makes it possible to reduce the software development costs.

Knowing that quality is the major criterion for software acceptance and success, we try in this work to find if there is a link between internal measurements made on OO components and three external quality factors: fault-proneness, corrective maintenance cost and reusability.

In this work, we obtain and exploit measurements collected throughout some softwares development cycles. Different machine learning algorithms are explored with regard to their capacities to produce predictive models. The predictability of each model is then evaluated and the internal measurements influencing the prediction of a quality factor will be identified.

Key words : software quality, estimation techniques, machine learning.

# Table des matières

# Liste des tableaux

# Liste des figures

# Liste des abréviations

| Acronyme | Description |
|---|---|
| AA | Apprentissage Automatique. |
| CRIM | Centre de Recherche Informatique de Montréal. |
| LALO | Langage d'Agents Logiciels Objets. |
| MLTOOLS | Machine Learning Tools. |
| RN | Réseaux de Neurones, «Neural Network». |
| RB | Réseaux Bayésiens. |
| SVM | Machine à Support de Vecteurs, « Support Vector Machines ». |
| WEKA | Waikato Environnement for Knowledge. |

# Remerciements

Je tiens à remercier vivement Mr Houari Sahraoui, mon directeur de recherche pour son appui et ses directives enrichissantes qui m'ont éclairé.

Je tiens à exprimer mes remerciements à Mr Hakim Lounis, mon codirecteur de recherche pour son aide et le temps précieux qu'il m'a accordé.

Sans leur aide, mon travail n'aurait pas été accompli.

Je tiens aussi à remercier infiniment Mr Nacer Izza qui a eu la patience de réviser mon manuscrit.

Finalement, je remercie toutes les personnes qui ont contribué directement ou indirectement à la réalisation de ce travail.

# Dédicaces

Il me tient à cœur de dédier ce travail à ma famille qui est loin de moi mais si présente dans mon cœur. Il serait sans doute long de les énumérer toutes et tous, néanmoins :

À ceux dont l'absence n'exclut pas leur présence dans mon cœur et dans mon esprit et à ceux dont l'existence est ma raison de vivre.

# Chapitre 1

# Problématique

## 1.1 Introduction et problématique

La concurrence dans le marché des logiciels impose une productivité de haute qualité et au moindre coût. La qualité d'un produit logiciel étant le critère majeur pour son acceptation et son succès.

Pour atteindre cet objectif, plusieurs recherches tentent d'utiliser des mesures tirées des réalisations précédentes afin d'établir des normes et des règles qui rendent les coûts et la qualité des logiciels plus abordables et plus fiables.
Ces mesures sont nommées « métriques ».

La disponibilité des métriques est un facteur essentiel pour couronner de succès la gestion du développement de logiciels. Quantitativement, elles sont là pour contrôler le développement des logiciels ainsi que leur qualité intrinsèque .
Ces métriques, source d'information avérée pour la prise de décision, sont nécessaires dans l'identification du niveau requis des ressources. C'est une base d'informations cruciales pour la prise de décision.
L'objectif de notre recherche est d'utiliser les algorithmes d'apprentissage (que nous définirons au chapitre 3) afin d'établir des règles qui nous permettent de prendre les bonnes décisions concernant les ressources à exploiter durant le cycle de vie du logiciel [44]. Le but est d'éviter les risques qui peuvent nuire aux facteurs de qualité du logiciel et améliorer la qualité du produit final.

Nous allons travailler sur 3 facteurs de mesure de qualités qui sont :

- Le degré de réutilisation de classes C++.
- La propension de classes C++ à engendrer des erreurs.
- L'effort pour la correction de composants ADA.

### 1.1.1 Le degré de réutilisation de classes C++

Au fur et à mesure que la complexité des programmes s'est accrue, il est apparu probant que la clef d'une bonne conception était la possibilité de réutiliser des éléments du programme. À travers le mécanisme d'héritage, l'approche orientée objet permet la réutilisation de composants (classes, objets, opérations).

La réutilisation permet de concevoir de nouveaux objets à partir d'objets existants. Ainsi, il est possible de spécialiser un objet pour des besoins spécifiques. On peut donc réutiliser les objets dans plusieurs applications. Cette opération permet d'accroître la vitesse de développement et la réduction des coûts sans oublier l'apport professionnel des concepteurs et développeurs.

Il faut noter que la réutilisation ne se limite pas uniquement à la duplication de code ou à des réadaptations. D'autres techniques sont aussi envisageables telles que le pré-processing et les bibliothèques de composants.
Le pré-processing est similaire à la duplication mais il permet également à l'application d'origine de bénéficier des correctifs apportés. Quant aux bibliothèques de composants, elles ont l'avantage de faire profiter tous les programmes des améliorations incluses dans la bibliothèque.

Certains attributs internes peuvent s'avérer d'excellents indicateurs sur la réutilisation de composants. Notre objectif est de trouver des relations directes entre ces attributs mesurables et le potentiel d'un composant à une seconde

utilisation. Dans notre étude nous allons vérifier trois hypothèses concernant l'existence de relation entre des attributs internes (qui sont le couplage, la complexité et l'héritage) et la réutilisabilité de certains composants orientés objets.

### 1.1.2 La propension de classes C++ à engendrer des erreurs

Le but ultime des développeurs est de fournir des produits exempts d'erreurs afin de réduire les efforts de correction. Aussi, trois hypothèses impliquant des liens entre des attributs internes (qui sont le couplage, la complexité et l'héritage) et la propension à engendrer des erreurs seront vérifiées.

Nous allons générer à partir des mesures de propriétés conceptuelles, des modèles de classification prédisant les composants qui sont susceptibles d'être erronés.

On a souvent remarqué qu'un petit nombre de composants logiciels sont responsables d'une multitude d'erreurs pendant la phase de développement de logiciel. Car dans le cas de la programmation orientée objet un composant peut constituer un élément de base pour créer d'autres composants logiciels(par la relation d'héritage). Si ce dernier est erroné alors le problème va se répercuter sur tous les autres composants.

Des recherche tentent de déterminer les modules qui sont susceptibles de contenir un nombre significatif d'erreurs. Dans le but d'atteindre les mêmes objectifs, nous allons bâtir des modèles quantitatifs qui prédisent lesquels des composants renferment la plus grande concentration d'erreurs. Une fois ces composants à grand risque identifiés le processus de développement de logiciel sera optimisé qualitativement afin d'en réduire les risques.

Nous proposerons 3 hypothèses: nous essayerons d'établir le lien entre des propriétés conceptuelles orientées objets (qui sont l'héritage, la cohésion et le couplage) et la propension à engendrer des erreurs, considéré comme un indicateur de qualité du logiciel.

Le génie logiciel accorde une grande importance à la maintenance des logiciels.

### 1.1.3 L'effort pour la correction de composants Ada

Plusieurs études ont démontré que 65 à 75% de l'effort total déployé durant le cycle de vie d'un logiciel l'est, en fait, pour sa maintenance. Cette phase influe considérablement les coûts de développement.

Il est juste de noter que la quantité d'effort à déployer pour maintenir un composant logiciel est inestimable. La difficulté de maintenance du logiciel augmente les coûts, rendant le produit moins attractif. Serait-ce dans ce cas possible d'évaluer la maintenance avant la livraison finale ?

Pour répondre à cette problématique, nous allons utiliser un historique de maintenance de certains composants écrits en langage Ada afin de vérifier la corrélation entre des mesures d'attributs internes et l'effort de maintenance. Nous allons exploiter des algorithmes d'apprentissage à des fins de vérification du lien causal entre ces mesures et l'effort déployé pour isoler et corriger un composant logiciel Ada.

## 1.2 Objectifs de la recherche

Sur la multitude de projets de développement de logiciel conçus chaque année, beaucoup ne satisfont pas les attentes des clients ou ne respectent pas les contraintes de budget ou de délai. De plus, des erreurs peuvent survenir après la livraison du logiciel, rendant la tâche de maintenance lourde et coûteuse. Conscients de ces difficultés et sachant que la plupart des activités entreprises pour le développement de logiciels sont de nature créative et rarement

automatisée, notre étude vise à démontrer l'existence de relations entre la façon de concevoir et de réaliser des composants logiciels avec certains facteurs décrivant la qualité du produit final.
Nous utiliserons les algorithmes d'apprentissages afin de prédire des attributs de qualité de produits logiciels, ceci, en exploitant les mesures de quelques attributs internes prélevés tout au long du développement de composants logiciels tel que la complexité, le couplage, la cohésion et l'héritage.

Il est évident que les codes réalisés par les développeurs renferment une mine de règles. En suivant de près les expériences déjà vécues, on peut déceler des relations entre la façon de produire et la qualité du produit final. Dans notre étude, nous allons baser notre recherche sur plusieurs échantillons de caractéristiques de composants logiciels orientés objets. Ces caractéristiques présentent différentes mesures effectuées sur des applications orientées objets ainsi que des mesures de qualité externes. On cherchera à trouver une relation de cause à effet entre ces mesures internes et les mesures de qualité suivantes :

- Réutilisabilité.
- Maintenabilité.
- Propension à engendrer des erreurs.

Aussi, des modèles de prédiction seront mis au point afin de donner aux développeurs les recommandations adéquates qui leur permettront dans le futur d'améliorer la qualité du produit final :

- Diminuer les coûts de maintenance.
- Augmenter la réutilisation des composants logiciels.
- Détecter les erreurs très tôt dans le cycle de vie de développement du logiciel.

- Gagner du temps dans le développement de logiciel grâce à la réutilisation.

Pour effectuer cette étude, nous disposons de peu de données ; ce qui va lui conférer une vision relativement restreinte. Ceci est du au fait que très peu d'entreprises effectuent un suivi rigoureux des projets informatiques. Dans plusieurs cas on peut constater le manque de documentation pertinente et récente accompagnant un code source. Ainsi, on retrouve des systèmes qui fonctionnent mais qui ne sont pas correctement documentés. Une autre raison est le manque de procédés de suivi de la maintenance.

## 1.3 Publications

En 2004, les travaux de recherche de ce mémoire ont conduit à la publication des deux articles suivants :

1. H. Lounis, L.Ait-Mehedine, « Machine Learning-Based Quality Predictive Models : Towards an Artificial Intelligence Making System ». In proceedings of the 16th International Conference on SE and KE, JUNE 2004, P 508-512.
2. H. Lounis, L. Ait-Mehedine, « Machine-Learning Techniques for SOFTWARE product quality Assessment ». In the proceedings of the 4th IEEE Internatioanl Conference on Quality Software, Sept. 2004, p 102-109.

## 1.4 Organisation du mémoire

Le présent document sera organisé comme suit :

Dans le deuxième chapitre nous allons aborder le thème de la qualité et des métriques orientées objets. Le troisième chapitre sera consacré aux différents algorithmes d'apprentissage que nous avons utilisés dans la génération de nos résultats. Le quatrième chapitre présente les problèmes de prédiction/estimation étudiés. Quant au cinquième chapitre, il présente les résultats de notre recherche. Enfin, le dernier chapitre sera dévolu à la conclusion générale et aux perspectives possibles de ce travail.

# CHAPITRE 2

# Qualité du logiciel et métriques orientées objets

## 2.1 Introduction

Si on arrive à prédire et à maîtriser la qualité, on sera capable d'atténuer les coûts en diminuant les risques de dysfonctionnement des logiciels et garantir ainsi une régularité de la production.

Dans ce chapitre, nous allons définir certains attributs de qualité parmi les plus importants. Par ailleurs, nous aborderons également des notions sur les mesures en génie logiciel et leur apport d'une façon générale.

## 2.2 Qualité du logiciel

Au niveau de l'entreprise, la qualité, qui concerne toutes les activités, tous les métiers et toutes les personnes, a pris une grande ampleur. Son importance s'étant accrue, un organisme international a édicté une série de normes à cet égard (Norme ISO 9126).

Nous pouvons affirmer que la qualité fait partie intégrante du processus de fabrication d'un logiciel. Selon Pressman, la qualité est définie comme étant la conformité à des besoins fonctionnels, des performances explicites, des standards professionnels de développement abondamment documentés ainsi

que des caractéristiques implicites attendues de tout logiciel développé suivant des normes professionnelles.

La définition objective de la qualité est donnée par Crosby qui écrit que *: « Les objectifs de qualité sont à atteindre par la prévention associée à un contrôle planifié et non par un constat après coup ».* Il affirme que la qualité s'obtient facilement d'autant qu'elle est gratuite car elle se mesure par rapport aux objectifs fixés au départ. Ainsi, pour l'obtenir à moindre coût il faut :

- Établir l'estimation des charges et délais en tenant compte de façon précise du niveau d'exigence requis pour chacun des critères de qualité retenus.
- Se doter de méthodes, procédures et outils qui vont nous permettre d'atteindre les objectifs de qualité que l'on se donne.

Quant à Glen Myers [53], il affirme que : *« tout programme est faux tant qu'on n'a pas prouvé le contraire ».* Son approche stipule que la qualité d'un logiciel n'est pas réduite à son exemption d'erreurs. Selon Myers, la recherche des erreurs dans un logiciel est une tâche difficile et qui exigerait, plus que toute autre activité du cycle de développement, des méthodes rigoureuses, des spécialistes formés et beaucoup de temps.

En génie logiciel, divers travaux ont mené à la définition de la qualité du logiciel en termes de facteurs qui dépendent, entre autres, du domaine de l'application et des outils utilisés. Il existe des critères de qualité qui permettent de définir différents types de qualité. Un développement peut être réalisé pour satisfaire tout ou partie de l'ensemble des critères.

Nous prendrons l'exemple de J.A.Mc Call [50] qui a identifié, dans une communication intitulée *Factors in Software Quality* plus de cinquante facteurs-candidats permettant d'exprimer la qualité.

Nous allons présenter les facteurs que Mc Call juge importants :

- L'adaptabilité mesure l'aptitude du logiciel à faciliter l'adjonction de nouvelles fonctionnalités ou la modification de fonctionnalités existantes (cet aspect couvre la correction des défauts provenant d'une mauvaise expression des besoins).
- La confidentialité mesure l'aptitude d'un logiciel à être protégé contre tout accès non autorisé.
- la correction mesure le degré de conformité par rapport aux spécifications.
- La couplabilité mesure l'aptitude d'un logiciel à être intégré dans un ensemble plus vaste.
- L'efficacité mesure l'aptitude d'un logiciel à minimiser la consommation des ressources qu'il utilise.
- L'ergonomie mesure l'aptitude d'un logiciel à être d'une utilisation agréable et facile pour l'utilisateur à qui il est destiné.
- La maintenabilité mesure l'aptitude d'un logiciel à faciliter la localisation et la correction d'erreurs résiduelles (il s'agit bien là de correction de défauts de non-conformité par rapport aux spécifications et non de défauts provenant d'une mauvaise expression des besoins).
- Portabilité : facilité avec laquelle un logiciel peut être transféré sous différents environnements matériels et logiciels.

- Réutilisabilité : aptitude d'un logiciel à être réutilisé, en entier ou en partie dans de nouvelles applications.
- Fiabilité(ou robustesse) : mesure l'aptitude du logiciel à conserver un comportement conforme aux besoins dans le cas d'événements imprévus.
- Testabilité : mesure l'aptitude d'un logiciel à faciliter la vérification de son comportement par rapport à des critères de test.

Ces critères de qualité sont des objectifs qu'un utilisateur va spécifier éventuellement dans l'expression de ses besoins. Notons que le choix d'une bonne méthode de développement devrait permettre la satisfaction des critères de qualité.

## 2.3 Métriques Orientées Objets

Sans métriques, il ne peut y avoir de politique de qualité. Les métriques sont très importantes pour contrôler, mesurer et fournir des indications sur la façon dont l'application est développée. De ce fait, de nombreuses propositions de métriques pour les systèmes orientés objets ont vu le jour. Parmi elles, celles de S.R. Chidamber et C. F. Kemerer dans « A Metrics Suite for Object Oriented Design » [60]. Cette proposition est en effet systématiquement citée dans les bibliographies des travaux récents sur les métriques orientées objets.

Il est important de souligner que c'est l'utilisation de la technologie des objets dans le développement industriel de logiciels qui a incité à développer des outils automatiques supportant des métriques. Ces outils sont essentiels pour l'utilisation des métriques produites sur une base régulière afin de surveiller et améliorer le logiciel [36].

Afin que les métriques soient activement et couramment utilisées, ces outils doivent être simples et faciles d'utilisation. La mise en œuvre des outils de collection de métriques peut de manière significative aider à améliorer les habilités du praticien à identifier, analyser, établir et améliorer les caractéristiques de qualité relatives à la conception du logiciel et à son implantation.

## 2.4 Les mesures en génie logiciel

En génie logiciel, l'utilisation des mesures est essentielle. Elles permettent l'évaluation des attributs spécifiques des composants logiciels et l'étude des dépendances entre les composants et leur environnement.

Kirby [39] soutient que les métriques ont contribué pleinement à l'accomplissement des grands évènements de l'Histoire comme les pyramides égyptiennes qui n'auraient jamais été édifiées sans elles.

Au dix-huitième siècle, la mesure a joué un grand rôle dans l'amélioration des technologies telles que les routes, les canaux, les ponts et la construction. Plus récemment, des personnalités importantes parmi lesquelles, Couloumb, Ampere, Faraday, Ohm, Oerested et Edison ont grandement contribué à l'établissement des disciplines du génie électrique et de l'électromagnétisme. La mesure tenait une place importante dans l'effort d'innovation entrepris par ces chercheurs.

Oman et Pfleeger [55] ont identifiés 6 raisons pour lesquelles « mesurer » est transitif :

- Mesurer pour le contrôle de projet : les mesures nous permettent d' évaluer les étapes dans un projet et de prédire le futur.

Rectifier le tir si nécessaire peut s'avérer salutaire afin d'atteindre nos objectifs.

- Mesurer pour la compréhension. En effet, certaines mesures nous permettent de mieux comprendre les activités de développement et de maintenance de logiciels. Nous pouvons ainsi évaluer la situation courante en établissant des points de base qui nous aident à fixer les objectifs pour les comportements futurs.
- Mesurer pour l'expérimentation qui est nécessaire en génie logiciel. Elle permet d'améliorer notre connaissance des normes de développement des logiciels, de mieux comprendre les effets des technologies diverses et d'identifier les champs nécessitant un champs d'amélioration plus soutenu. Les mesures jouent un rôle important dans l'expérimentation et nous permettent de tester des hypothèses et d'en bâtir de nouvelles.
- Mesurer pour l'amélioration du processus : la mesure nous permet de mieux évaluer le processus et comprendre la portée des changements aménagés et procéder ainsi à une amélioration constante.
- Mesurer pour l'amélioration du produit : la mesure offre une vision sur les façons dont les processus, les produits, les ressources, les méthodes et les technologies de développement de logiciels s'apparentent. Les mesures nous aident à répondre à des questions sur l'efficacité des techniques et des outils, la productivité des activités de développement, la qualité du produit, etc. Cela influe grandement sur le développement du produit.

- Mesurer pour évaluer des prévisions : aux nouvelles activités, il faut être à même de prévoir l'effort requis ainsi que les coûts de développement et autres. Les mesures nous permettent d'avoir un point de repère afin d'entreprendre des prévisions sur ces activités. Le fait d'attendre simplement la fin du projet et ensuite de mesurer les attributs de coûts et de temps est clairement inacceptable.

Dans le livre : « Software metrics :A Rigourous & Practical approach » de Fenton et Pfleeger, 1997, on peut identifier une septième mesure, celle de l'évaluation :

- Mesurer pour l'évaluation : C'est très utile afin de pouvoir comprendre les enjeux actuels ou ceux du passé.

Dans le passé, la recherche dans le domaine de la mesure de logiciel a souffert d'un manque de : **1-** terminologie standardisée et **2-** d'un formalisme pour définir des mesures non ambiguës et totalement opérationnelles, c'est-à-dire une manière qui n'exige aucune interprétation supplémentaire par l'utilisateur de la mesure [9], [12].

Cela a eu comme conséquence d'entraver sévèrement le développement de logiciel avec la qualité d'accessibilité et de souplesse prédits. Pour remédier à cette situation, il est nécessaire d'arriver à un consensus sur la terminologie et la définition d'un formalisme clair qui aura à exprimer les mesures d'un logiciel.

Pour représenter les métriques orientées objets que nous avons utilisé, on a élaboré la terminologie et le formalisme suivants :

### 2.4.1 Terminologie et formalisme

Pour la définition des mesures orientées-objets établies pour la mise en place de nos trois expérimentations, nous avons exploité la terminologie utilisée dans les travaux de [35], [47].

#### a) Système, classe et relations d'héritage

Chaque système orienté-objet $S$ est une collection de classes $C$. Des relations d'héritage entre classes peuvent exister, tel que pour chaque classe $c \in C$ *on a :*

- $Parents(c) \subset C$ : l'ensemble des classes parents de c
- $Children(c) \subset C$ : l'ensemble des classes enfants de c
- $Ancestors(c) \subset C$ : l'ensemble des classes ancêtres de c
- $Descendants(c) \subset C$ : l'ensemble des classes descendantes de c

En C++, une classe c peut déclarer une classe d comme amie si d peut accéder aux éléments non publics de c.

Pour une certaine classe c, on doit spécifier ses classes amies et la liste des classes qui déclarent la classe **c** comme classe amie. Ainsi, on définit :

- $Friends(c) \subset C$ : l'ensemble des classes amies de c
- $Friends'(c) \subset C$ : l'ensemble des classes qui déclarent la classe c comme amie
- $Others(c) \subset C$ : l'ensemble des autres classes de c

Où

$$Others(c) = C \setminus (Ancestors(c) \cup Descendants(c) \cup Friends(c) \cup Friends'(c) \cup \{c\})$$

**b) Méthodes**

Une classe **c** contient un ensemble de méthodes M (c). Une méthode peut être virtuelle ou non virtuelle et aussi héritée, surchargée ou bien nouvellement définie. Ainsi, M(c) peut comprendre un ensemble de méthodes déclarées $M_D(c) \subseteq M(c)$ et un ensemble de méthodes implémentées.

$$M(c) = M_D(c) \cup M_I(c) \text{ et } M_D(c) \cap M_I(c) = \Phi$$

Où

- $M_D(c)$ : l'ensemble des méthodes déclarées de c
- $M_I(c)$ : l'ensemble des méthodes implémentées de c

On définit aussi :

- $M_{INH}(c) \subseteq M(c)$ : l'ensemble des méthodes héritées de c
- $M_{OVR}(c) \subseteq M(c)$ : l'ensemble des méthodes surchargées de c
- $M_{NEW}(c) \subseteq M(c)$ : l'ensemble des méthodes nouvelles, non héritées et non-surchargées de c.

Où

$$M(c) = M_{INH}(c) \cup M_{OVR}(c) \cup M_{NEW}(c)$$

- $M_{PUB}(c) \subseteq M(c)$ : l'ensemble des méthodes publiques
- $M_{NPUB}(c) \subseteq M(c)$ : l'ensemble des méthodes privées

En C++, il peut y avoir des méthodes publiques, protégées et privées. Aussi, peut-on diviser $M(c)$ en méthodes publiques et non publiques d'où :

$$M(c)=M_{Pub}(c)\cup M_{npub}$$

Pour chaque méthode d'une classe, il y a un ensemble de paramètres Par(m). Pour mesurer la cohésion et le couplage d'une classe, il est nécessaire de définir l'ensemble des méthodes invoquées par m∈ M(c) ainsi que la fréquence de ces invocations. Les invocations de méthodes peuvent être statiques ( Méthode qui ne peut accéder qu'aux données statiques d'un objet.) ou dynamiques (Méthode qui peut changer dynamiquement dans le temps.). Par conséquent, pour chaque méthode m∈ M(c) l'ensemble suivant est défini:

- $SIM(m)$ : l'ensemble des méthodes statiques invoquées de m.
- $SIM^{*}(m)$ : l'ensemble des méthodes indirectement invoquées.
- $PIM(m)$ : l'ensemble des méthodes polymorphiques (Méthode qui réfère a des objets de plus d'une classe/d'un type.) invoquées de m.

**c) Attributs**

Les classes ont des attributs qui peuvent être hérités ou nouvellement définis. Pour chaque classe $c \in C$ on suppose que A(c) est l'ensemble des attributs de la classe c.

- $A(c) = A_D(c) \cup A_I(c)$ où
  - $A_D(c)$ est l'ensemble des attributs déclarés dans la classe c.
  - $A_I(c)$ est l'ensemble des attributs implémentés dans la classe c.

Les méthodes peuvent référencer des attributs. Il est suffisant de considérer le type statique de l'objet pour lequel un attribut est référence car les références d'un attribut ne sont pas déterminées dynamiquement. On définit :

- $AR(m)$ est l'ensemble des attributs référencés par la méthode m, $m \in M(c)$
- $NAR(m,a)$ est le nombre de références de la méthode m à l'attribut a

**d) Prédicats, interaction, ensembles dérivés**

Pour mesurer le couplage entre les classes c, d $\in C$ *on définit le prédicat suivant :*

- uses(c,d) : une classe c est couplée à une autre classe d s'il existe une quelconque relation (utilisation, agrégation, héritage) entre c et d dans un sens ou dans un autre.

Et pour mesurer certaines interactions entre les classes c, d $\in C$ on définit :

- ACA(c, d) est le nombre des interactions[1] actuelles entre classe-attribut à partir de la classe c vers la classe d.
- ACM(c, d) est le nombre des interactions actuelles entre classe-méthode à partir de la classe c vers la classe d.
- AMM(c, d) est le nombre des interactions actuelles entre méthode-méthode à partir de la classe c vers la classe d.

[1] Échanges d'informations.

## 2.6 Classification des métriques par auteurs

Nous allons définir l'ensemble des mesures utilisées au cours de nos expériences pour générer les résultats du chapitre 5.

### 2.6.1 Métriques proposées par Chidamber et Keremer [17], [18]

Ces métriques visent à mesurer la complexité de la conception des classes :

1. Weighted Methods per Class (WMC) : c'est la somme pondérée des complexités des méthodes d'une classe. Le nombre et la complexité des méthodes d'une classe est un bon indicateur du temps nécessaire pour développer et maintenir une classe. Plus le nombre de méthodes dans une classe est important, plus grand sera l'impact sur les classes enfants via l'héritage. Cette métrique sert à :
    - Prédire le temps et l'effort nécessaire au développement et à la maintenance d'une classe.
    - Évaluer la réutilisabilité : les classes avec un grand nombre de méthodes complexes sont en général moins réutilisables.

2. Depth of Inheritance Tree (DIT) : c'est le nombre maximum de classes ancêtres de la classe pour atteindre la (une) racine.

$$DIT(c) = \begin{cases} 0, si\ parents(c) = 0 \\ 1 + \max\{DIT(c') : c' \in Parents(c)\}, \text{ sinon} \end{cases}$$

   Représente un reflet de la complexité via la portée des ancêtres. Plus une classe se situe profondément dans l'arbre d'héritage, plus le nombre de méthodes et de comportements dont elle hérite sera grand. Plus un arbre est profond, plus la conception est complexe. Elle sert à évaluer :

- La complexité de la classe. Le comportement de la classe étant plus difficile à comprendre lorsque le nombre de méthodes héritées croît.
- La complexité globale. La conception est plus complexe puisqu'il y a plus de classes et de méthodes à développer.
- La réutilisabilité de la classe. Plus une classe est loin dans la hiérarchie, moins elle est générique.

3. Number of children (NOC) : c'est le nombre de classes immédiatement sous la classe dans l'arbre d'héritage (le nombre de sous-classes directes de cette classe).

   $NOC(c) = |Children(c)|$

   Représente un reflet de l'impact potentiel d'une classe sur ses descendants. Plus le nombre d'enfants est élevé, plus le risque d'une utilisation inadéquate du mécanisme de l'héritage est grand et nécessitera davantage d'efforts de test. Elle sert à évaluer :

   - La réutilisabilité : une classe ayant de nombreux enfants étant très générique.
   - La mauvaise abstraction : une classe ayant de nombreuses classes dérivées a une plus grande probabilité d'être improprement abstraite.
   - L'influence sur le système et sur l'effort de test : une classe ayant de nombreuses classes dérivées a un impact fort sur la hiérarchie de classes. Un effort de tests particulier devra lui être appliqué.

4. Coupling between object classes CBO :

   $CBO(c) = |\{d \in C \setminus \{c\} : uses(c,d) \vee uses(d,c)\}|$

Cette métrique compte le nombre de classes avec lesquelles une classe est couplée. Une classe C1 est couplée à une autre classe C2 s'il existe une quelconque relation (utilisation, agrégation, héritage) entre C1 et C2 dans un sens ou dans un autre.

Un fort couplage se fait au détriment de la modularité et empêche la réutilisation. Plus le CBO est faible pour une classe, davantage cette classe est facilement réutilisable.

La métrique CBO sert à :

- Évaluer la modularité et la réutilisabilité : Un couplage excessif entre classes se fait au détriment de la modularité et empêche la réutilisation. Plus le CBO est faible pour une classe, plus cette classe est facilement réutilisable.
- Évaluer de l'effort de maintenance : Plus une classe est couplée à d'autres classes, plus une modification de cette classe peut affecter d'autres parties du système.
- Évaluer la testabilité : Une mesure du couplage est importante pour déterminer la complexité du test des diverses parties d'un logiciel. Des efforts de test d'une classe accrus sont nécessaires et proportionnels lorsque la mesure du CBO est élevée.

5. Response For a Class (RFC) : RFC vaut la cardinalité de l'ensemble de réponse de la classe. L'ensemble de réponse d'une classe est celui des méthodes qui peuvent être directement appelés lors de l'exécution de n'importe quelle méthode de cette classe (en réponse à un message). C'est un reflet du niveau de communication potentiel entre une classe et les autres. La complexité d'une classe est d'autant plus élevée qu'elle peut appeler un grand nombre de méthodes.

   La métrique RFC sert à :

   - Évaluer la testabilité : plus une classe invoque des méthodes de diverses origines, plus elle est compliquée à comprendre.

- Évaluer la complexité : plus une classe invoque des méthodes, plus elle est complexe.
- Évaluer l'effort de test : Une classe complexe requiert des efforts de test plus importants.

6. $LCOM_2(c)$ : Cette métrique est utile pour l'estimation du degré de cohésion dans un système. Elle représente le nombre de méthodes, prises deux à deux, ne partageant pas des instances de variables de la classe moins le nombre de méthodes prises deux à deux partageant des instances de variables de la classe. Si cette valeur est négative, LCOM2 est fixée à zéro.

Cette métrique sert à :

- Évaluer l'encapsulation : Une classe cohérente promouvant celle-ci.
- Évaluer la complexité : une classe disparate augmentant la probabilité d'erreur durant le développement.
- Évaluer les défauts de conception de classes : Si une classe est peu cohérente, il serait préférable de faire éclater cette classe en plusieurs autres classes plus cohérentes.

### 2.6.2 Métrique proposées par Li et Henry [42]

De l'ensemble des métriques définis par ses auteurs, nous avons exploités les deux métriques de couplage suivantes :

1. Message Passing Coupling (MPC) : défini comme étant le nombre de messages envoyés dans une classe.

$$MPC(c) = \sum_{m \in M_I(c)} \sum_{m' \in SIM(m) | M_I(c)} NSI(m, m')$$

2. Data abstraction coupling DAC : cette métrique permet d'évaluer le couplage interne d'une classe avec d'autres classes..

$$DAC(c) = \sum_{d \in C, d \neq c} ACA(c,d)$$

### 2.6.3 Métriques proposées par Briand et Al [9], [11]

Toutes les mesures présentés ici sont des mesures de couplage, excepté la mesure « Coh » qui est une mesure de cohésion.

1. ACAIC : Ancestors Class-Attribute Import Coupling, c'est le nombre d'interactions class-attribute entre une classe et ses ancêtres.

$$ACAIC(c) = \sum_{d \in Ancestors(c)} ACA(c,d)$$

2. IFCAIC : Inverse Friend Class-Attribute Import Coupling. c'est le nombre d'interactions class-attribute entre une classe et les classes avec lesquelles elle est amie.

$$IFCAIC(c) = \sum_{d \in Friends^{-1}(c)} ACA(c,d)$$

3. OCAIC : Others Class-Attribute Import Coupling, c'est le nombre d'interactions 'class-attribute' entre une classe et les autres classes (qui ne sont ni 'friends', ni 'inverse friends', ni descendants, ni ancêtres)

$$OCAIC(c) = \sum_{d \in Others(c)} ACA(c,d)$$

4. FCAEC : Friends Class Export Coupling. C'est le nombre de classes amies avec lesquelles une classe a un couplage d'exportation.

$$FCAEC(c) = \sum_{d \in Friends(c)} ACA(d,c)$$

5. DCAEC : Descendants Class-Attribute export coupling, c'est le nombre de sous-classes avec lesquelles une classe a un couplage de type exportation (EC).

$$DCAEC(c) = \sum_{d \in Descendants(c)} ACA(d,c)$$

6. OCAEC : Others Class-Attribute Export Coupling, c'est le nombre de classes avec lesquelles une classe a un couplage de type exportation autres que les super-classes ou les sous-classes.

$$OCAEC(c) = \sum_{d \in Others(c)} ACA(d,c)$$

7. IFCMIC : Inverse Friend Class-Method Import Coupling. c'est le nombre d'interactions 'class-method' entre une classe et les classes avec lesquelles elle est amie.

$$IFCMIC(c) = \sum_{d \in Friends^{-1}(c)} ACM(c,d)$$

8. ACMIC : Ancestors Class-Method import coupling, c'est le nombre de classes parentes avec lesquelles une classe a un couplage de type importation.

$$ACMIC(c) = \sum_{d \in Ancestors(c)} ACM(c,d)$$

9. OCMIC : Others Class-Method Import Coupling, c'est le nombre de classes avec lesquelles une classe a un couplage de type importation, autres que les super-classes ou les sous-classes.

$$OCMIC(c) = \sum_{d \in Others(c)} ACM(c,d)$$

10. FCMEC : Friends Class-Method Export Coupling. C'est le nombre de classes parentes avec lesquelles une classe a un couplage d'importation.

$$FCMEC(c) = \sum_{d \in Friends(c)} ACM(c,d)$$

11. DCMEC : Descendants Class-Method export coupling, c'est le nombre de sous-classes avec lesquelles une classe a un couplage de type exportation.

$$DCMEC(c) = \sum_{d \in Descendants(c)} ACM(c,d)$$

12. OCMEC : Others Class-Method Export Coupling, c'est le nombre de classes avec lesquelles une classe a un couplage de type exportation autres que les super-classes ou les sous-classes.

$$OCMEC(c) = \sum_{d \in Others(c)} ACM(d,c)$$

13. OMMIC : Others Method-Method Import Coupling, c'est le nombre de classes avec lesquelles une classe a un couplage de type importation autres que les super-classes
ou les sous-classes.

$$OMMIC(c) = \sum_{d \in Others(c)} AMM(c,d)$$

14. IFMMIC : Inverse Friend Method-Method Import Coupling. c'est le nombre d'interactions 'method-method' entre une classe et les classes avec lesquelles elle est amie.

$$IFMMIC(c) = \sum_{d \in Friends^{-1}(c)} AMM(c,d)$$

15. AMMIC : Ancestors Method-Method Import Coupling, c'est le nombre de classes parentes avec lesquelles une classe a un couplage d'importation.

$$AMMIC(c) = \sum_{d \in Ancestors(c)} AMM(c,d)$$

16. OMMEC : Others Method-Method Export Coupling, c'est le nombre de classes avec lesquelles une classe a un couplage de type exportation autres que les super-classes ou les sous-classes.

$$OMMEC(c) = \sum_{d \in Others(c)} AMM(d,c)$$

17. DMMEC : Others Method-Method Export Coupling, c'est le nombre de classes avec lesquelles une classe a un couplage de type exportation, autres que les super-classes ou les sous-classes.

$$DMMEC(c) = \sum_{d \in Descendants(c)} AMM(d,c)$$

18. FMMEC : Friends Method-Method Export Coupling. c'est le nombre de classes amies avec lesquelles une classe a un couplage de type exportation.

$$FMMEC(c) = \sum_{d \in Friends(c)} AMM(d,c)$$

19. Lack of cohesion in methods (Coh) : C'est une variation de LCOM5 (voir

$$Coh(c) = \frac{\sum_{a \in A_I(c)} \left| \{ m : m \in M_I(c) \wedge a \in AR(m) \} \right|}{\left| M_I(c) \right| - A_I(c)}$$

### 2.6.4 Métriques proposées par Lee et al [40]

Nous avons utilisé une mesure de cohésion nommé « ICH » et trois mesures de couplages ( ICH, NIH-ICP, IH-ICP) :

1. Information flow based cohesion ICH(c): c'est le nombre d'invocations des autres méthodes de même classe

$$ICH^c(m) = \sum_{m' \in M_{NEW}(c) \cup M_{OVR}(c)} \left(1 + \left|Par(m')\right|\right) NPI(m,m')$$

$$ICH = \sum_{m \in M_I(c)} ICH^c(m)$$

2. Information based coupling ICP : Cette mesure a été introduite dans le but de mesurer le flux d'information entre méthodes.

$$ICP(c) = \sum_{m \in M_I(c)} ICP^c(m)$$

$$ICP^c(c) = \sum_{m' \in PIM(m) \setminus (M_{New}(c) \cup M_{OVR}(c))} (1 + \left|Par(m')\right|) NPI(m,m')$$

3. Information flow based non-inheritance coupling NIH-ICP(c) : comme ICP en comptant uniquement les invocations de méthodes non héritées.

$$NIH - ICP(m) = \sum_{m \in M_I(c)} NIH - ICP^c(m)$$

$$NIH-ICP^{c}(m)=\sum_{m'\in PIM(m)\cap\left(\bigcup_{c'\in Ancestors(c)}M'(c)\right)}(1+|Par(m')|)NPI(m,m')$$

4. Information flow based inheritance coupling IH-ICP(c): comme ICP en comptant uniquement les invocations de méthodes héritées.

$$IH-ICP(m)=\sum_{m\in M_I(c)}IH-ICP^{c}(m)$$

$$IH-ICP^{c}(m)=\sum_{m'\in PIM(m)\cap\left(\bigcup_{c'\in C\setminus\{\{c\}\cup Ancestors(c)\}}M(c')\right)}(1+|Par(m')|)NPI(m,m')$$

### 2.6.5 Métriques proposées par Lorenz et Kidd [46]

Voici l'ensemble des métriques appartenant a ses auteurs, utilisées dans nos recherches:

1. Number of methods overridden (NMO): C'est le nombre de méthodes d'une classe qui surdéfinissent une méthode héritée d'une super classe.

   $NMO(c)=|M_{OVR}(c)|$

2. Number of method added, new methods (NMA): Nombre de nouvelles méthodes ajoutée dans une classe( non sur-définis et non héritée)

   $NMA(c)=|M_{NEW}(c)|$

3. Number of method inherited (NMI): Le nombre de méthodes dans une classe, que la classe a héritée d'une classe ancêtre mais qu'elle n'a pas sur-définis.

   $NMI(c)=|M_{INH}(c)|$

4. Number of parents (NOP): Le nombre de classes qui héritent directement d'une classe donnée.

   $NOP(c)=|Parents(c)|$

5. Spécialisation index (SIX): Index de spécialisation définit par la formule suivante :

$$SIX(c) = \frac{NMO(c).DIT(c)}{|M(c)|}$$

### 2.6.6 Métriques proposées par Hitz et Montazeri [32], [33]

Nous avons utilisé les trois mesures de cohésion suivantes :

1. Connectivity (Co) : Si G est le graphe dont les sommets sont des méthodeset un arc xy correspond à l'appel de y par x, Co est approximativement le rapport entre le nombre d'arcs existants et le nombre d'arcs possibles [24] :

$$Co(c) = 2\frac{|E_c| - (|V_c| - 1)}{(|V_c| - 1)(|V_c| - 2)}$$

2. LCOM3 : Soit $G_c = (V_c, E_c)$ un graphe unidirectionnel où $V_c = M_I(c)$ $LCOM_3(c)$ est le nombre de composants connectés de $G_c$ avec :

$$E_c(c) = \{\ [m_1, m_2] : m_1, m_2 \in V_c(c) \wedge AR(m_1) \cap AR(m_2) \cap A_I(c) \neq \Phi \\ \vee\ m_1 \in SIM(m_2) \vee m_2 \in SIM(m_1)\ \}$$

3. LCOM4 :définie comme étant le nombre de paires de méthodes dans une classe n'ayant pas de références communes d'attribut, $|P|$, moins le nombre de paires des methodes similaires $|Q|$.

$$LCOM_4(c) = \begin{cases} 0, si\ |P| < |Q| \\ |P| - |Q|, \text{ sinon} \end{cases} \quad \text{où } P = \begin{cases} 0, si\ AR(m) = 0\ \forall\ m \in M_I(c) \\ condition\ de\ LCOM_1(c), \text{ sinon} \end{cases}$$

$$Q = E_c de\ LCOM_2(c)$$

### 2.6.7 Métriques proposées par Bieman et Kang [14]

Deux métriques de cohésion ont exploites de l'ensemble des mesures de Bieman et Kang, nous citons :

1. Tight class cohesion TCC: Cette mesure est définis comme étant le pourcentage des pairs de méthodes public, d'une classe donné, qui sont connectées. C'est à dire les paires de méthode publics qui utilisent, directement ou indirectement, des attributs en commun.

$$TCC(c) = \frac{\left|\left\{\left[m_1, m_2\right] : m_1, m_2 \in M_I(c) \cap M_{Pub}(c) \wedge m_1 \neq m_2 \wedge cau(m_1, m_2)\right\}\right|}{\left|M_I(c) \cap M_{Pub}(c)\right|\left(\left|M_I(c) \cap M_{Pub}(c)\right| - 1\right)}$$

$$cau(m_1, m_2) = \left(\bigcup_{m \in \{m_1\} \cup SIM^*(m_1)} AR(m)\right) \cap \left(\bigcup_{m \in \{m_2\} \cup SIM^*(m_2)} AR(m)\right) \cap A_I(c) \neq 0$$

2. Lose class cohesion (LCC): La mesure LCC représente le pourcentage des pairs de méthodes public, d'une classe donné, qui sont directement ou indirectement connectées.

$$LCC(c) = \frac{\left|\left\{\left[m_1, m_2\right] : m_1, m_2 \in M_I(c) \cap M_{Pub}(c) \wedge m_1 \neq m_2 \wedge cau^*(m_1, m_2)\right\}\right|}{\left|M_I(c) \cap M_{Pub}(c)\right|\left(\left|M_I(c) \cap M_{Pub}(c)\right| - 1\right)}$$

### 2.6.8 Métriques proposées par Tegarden et Al [63]

Soit c ∈ C. On définit :

1. Class to leaf depth (CLD) : c'est le nombre maximum de niveaux dans la hiérarchie au-dessous de la classe.

$$CLD(c) = \begin{cases} 0, si\ D\text{escendants}(c) = 0 \\ \max\{DIT(c') - DIT(c) : c' \in Descendants(c)\}, \text{ sinon} \end{cases}$$

2. Number of ancestors (NOA) : c'est le nombre de classes distinctes desquelles la classe hérite. S'il n'y a pas d'héritage multiple, cette mesure est la même que DIT.

$$NOA(c) = |Ancestors(c)|$$

3. Number of descendants (NOD): C'est le nombre de classes qui héritent directement ou indirectement d'une classe donnée.

$$NOD(c) = |Descendants(c)|$$

4. Number of inherited methods (OIM) : mesure le nombre de méthodes qui sont utilisées dans la classe courante mais définies dans des classes ancêtres

### 2.6.9 Métriques proposées par Henderson-Sellers [30], [31]

Nous présentons ci-dessous les métriques (celles utilisées dans nos expériences.) de Henderson [28] qui classifie les mesures en deux catégories :

- Mesure interne de classe.
- Mesure au niveau système.

1. Mesure interne de classe (class internal measure) : il s'agit d'une mesure conçue à l'interne de la classe. Peut s'exprimer par le seul moyen d'entités appartenant au produit. Cette catégorie de mesure évalue la taille d'une classe et sa complexité. Il faut noter que les métriques RFC et WMC présentée précédemment font partie de cette catégorie de mesures.

- Number of Polymorphic Methods (NOPM): c'est le nombre de méthodes polymorphiques (y compris surchargées).
- Number of local methods NOM: nombre de méthodes localement définies dans une classe(hors méthodes héritées).
- Class Interface Size (CIS) : c'est le nombre de méthodes publiques (globales) dans une classe
- Number Of Abstract Data Types (NAD) : c'est le nombre d'objets définis par l'utilisateur, utilisés comme attributs dans une classe.
- Class Size in Bytes (CSB) : c'est la taille, en octets, des objets qui seront crées dans une déclaration de classe. C'est la somme de la taille de tous les attributs déclarés dans une classe.
- Number of Public Attributes (NPA) :c'est le nombre des attributs déclarés comme publics dans une classe.
- Number Of Parameters Per Method (NPM) : c'est la moyenne du nombre de paramètres par méthode dans une classe. Calculée en sommant le nombre de paramètres de toutes les méthodes divisé par le nombre de méthodes dans une classe.
- Number of Reference Attributes (NRA) : c'est le nombre de pointeurs et références utilisés comme attributs dans une classe.

2. Mesures au niveau système (system level metrics): cette catégorie regroupe les mesures de classe interne et externe.

- Average Width of Inheritance (AWI) : c'est le nombre moyen de descendants par classe dans un système.
- Average Number of Ancestors (ANA) : c'est le nombre moyen de classes à partir desquelles une classe hérite de l'information

- Number of Leaf Classes (NLC) : c'est le nombre de classes feuilles dans la hiérarchie du système.
- System Size in Classes (DSC) : c'est le nombre total de classes dans un système.
- Number Of Hierarchies (NOH) : c'est le nombre de hiérarchies d'une classe dans un système.
- Number of Independent Classes (NIC) : c'est le nombre de classes (solitaires) qui ne sont pas héritées par d'autres classes.
- Number of Single Inheritance (NSI) : c'est le nombre de classes ou sous-classes qui utilisent un héritage unique.
- Average Inheritance Depth (AID) : c'est la profondeur moyenne de l'héritage de classes dans un système.
- Number of Multiple Inheritance (NMI) : c'est le nombre de classes qui utilisent un héritage multiple.
- Number of inherited methods(OIM) : mesure le nombre de méthodes qui sont utilisées dans la classe courante mais définies dans les classes ancêtres.

Deux métriques pour l'estimation de la cohésion appartenant a ce même auteur ont été exploitée :

- $LCOM_1(c)$ : Nombre de méthodes, prises deux à deux, ne partageant pas des instances de variables de la classe.

$$LCOM_1(c) = \left|\left\{\left[m_1, m_2\right] : m_1, m_2 \in M_I(c) \wedge m_1 \neq m_2 \wedge AR(m_1) \cap AR(m_2) \cap A_I(c) = \Phi\right\}\right|$$

- $LCOM_5(c)$ : C'est une autre mesure qui permet de calculer le degré de cohésion d'une classe en prenant en considération l'ensembles des méthodes qui font référence à un certains attributs de la classe.

$$LCOM_5(c) = \frac{|M_I(c)| - \frac{1}{A_I(c)} \sum_{a \in A_I(c)} |\{m : m \in M_I(c) \wedge a \in AR(m)\}|}{|M_I(c)| - 1}$$

## 2.7 Modèles prédictifs

Comme nous avons pu le constater, il existe un très grand nombre de métriques qui ont été définies afin de permettre un meilleur contrôle du processus de production de logiciel. C'est grâce à ces métriques que l'on pourra être en mesure de juger de la qualité d'un logiciel. Les métriques sont utilisées très tôt dans le cycle de vie pour voir si on est sur la bonne voie dans le développement.

Les qualités internes ainsi mesurées influencent les caractéristiques de qualité externes du logiciel développé telles que sa fiabilité, sa maintenabilité, etc. L'ISO a défini dans sa norme 9126 un modèle liant les qualités internes d'un logiciel à ses qualités externes [1], ainsi qu'un ensemble de mesures les capturant (Voir FIG. 2.1).

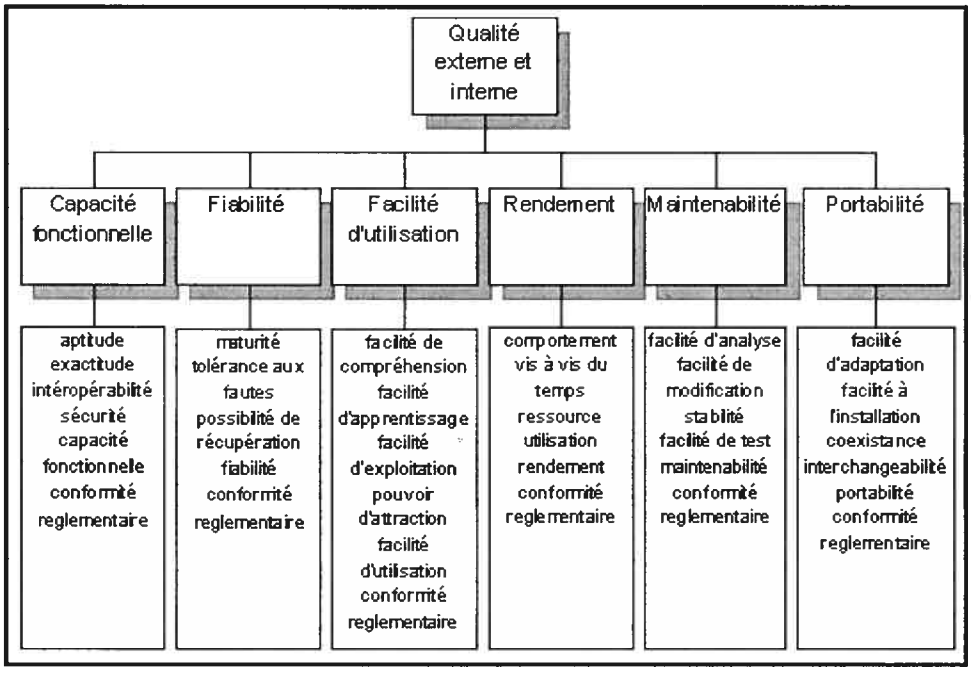


**FIG. 2.1: Le modèle de qualité du logiciel de la norme ISO 9126**

Ce modèle indique un lien de causalité entre des caractéristiques internes et des caractéristiques externes d'un produit, mais ne permet malheureusement pas de les prédire. C'est pour cette raison que les modèles de prédiction sont utilisés afin de capturer efficacement la relation entre les attributs internes d'un logiciel et les caractéristiques de qualité. En d'autre termes nous pouvons dire que ces derniers permettent de prédire les qualités externes en se basant sur les métriques logicielles (attributs internes).

L'objectif est donc de construire des modèles prédictifs de la qualité de logiciels, en se basant sur un ensemble d'apprentissage qui possède la même structure : un ensemble de paires attribut-valeur décrivant l'exemple, dans notre cas les différentes mesures internes du logiciels (exemple : mesures de couplage). Un attribut particulier (généralement le dernier attribut) représente la

classe de l'exemple. Si on estime la propension du logiciel à engendrer des erreurs , alors cet attribut indique le fait d'avoir relevé ou non des erreurs.
L'objectif de notre recherche consiste alors à déterminer un modèle qui permet de prédire correctement la qualité d'un logiciel et cela en se basant sur les valeurs de mesures internes d'une classe C++,.
Pour construire de tels modèles, nous utiliserons les algorithmes d'apprentissage (voir chapitre3), et pour les valider nous utiliserons certaines techniques que nous présentons dans la partie 5.3.

## 2.8 Travaux reliés

Les liens entre mesures internes et attributs de qualité ont fait l'objet de plusieurs recherches. D'autres travaux ont voulu prouver que l'utilisation des algorithmes d'apprentissage constitue une aide significative pour construire des modèles d'évaluation de la qualité des logiciels.

Nous allons résumer certains travaux ci-dessous:

En 1990, Poter and Selby [4] dans l'article « Empirically-guided software development using metric-based classification trees » furent les premiers à utiliser un algorithme d'apprentissage pour construire automatiquement un modèle de prédiction de la qualité de logiciel. Pour ce faire, ils ont utilisé le ID3, qui est un algorithme de classification, pour identifier les mesures internes qui constituent les meilleurs prédicteurs des erreurs d'interface que l'on peut rencontrer durant la phase de maintenance.

En 1991, Gianluigi Caldiera et Victor R. Basili [15], publient l'article « Identifying and qualifying reusable software components » dans lequel on retrouve des études de cas qui mettent en évidence que les techniques

expérimentales contribuent à la réduction de la quantité de code à évaluer pour identifier les composants réutilisables. Les chercheurs affirment que la mise en application de mesures de logiciels peut automatiser l'extraction de composants réutilisables à partir de systèmes existants et réduire le code que les experts devraient analyser.

En 1997, L. Etzkorn et C. G. Davis écrivent un texte intitulé « Automatically identifying reusable OO legacy code » [26] où ils affirment que la qualité de réutilisabilité ne figure pas dans la majorité des systèmes orientés-objets. Ils soutiennent que les commentaires et identificateurs refléteraient mieux les concepts humains, mieux en tous les cas que les algorithmes; d'où l'utilité d'outils basés sur la compréhension sémantique du code pour améliorer la réutilisabilité.

Dans un article au titre révélateur, « Analyzing and Measuring Reusability in Object-Oriented Designs » [54] publié en 1997, M. W. Price, S. Steven et A. Demurjian présentent une technique d'analyse et de mesure de la réutilisabilité de la conception orientée-objet. Ils soumettent par là même l'idée d'incorporer des mesures dans l'environnement de conception et de développement .

Parallèlement, et toujours en 1997, Briand, Devanbu et Melo [10] publient l'article « An Investigation Into Coupling Measures for C++ » dans lequel ils définissent un ensemble de métriques de couplage de systèmes orientés-objets au niveau de la conception. Les auteurs du texte ont présenté en primeur vingt-quatre métriques de couplage validées empiriquement en analysant la relation de cause à effet avec la probabilité de détection d'erreurs dans les classes. Notons que ces métriques ont été reprises dans notre étude.

Quant à V. R. Basili, Condon, K. E. Emam, R. B. Hendrick et W. Melo [5], ils traitent dans leur article intitulé « Characterizing And Modeling The Cost Of Rework in a Library Of Reusable Software Components », 1997, de la réutilisation de composants logiciels. Dans leur étude, ils affirment que les mesures de certains attributs internes permettent de prédire les efforts de maintenance. Pour corroborer leur thèse, ils présentent des modèles de prédiction des coûts de maintenance. Ces modèles ont été utilisés par des chefs de projets pour l'allocation de ressources aux activités de maintenance corrective. Les travaux de cette recherche ont été mis au point en collectant les informations sur les activités de la maintenance corrective pour la « Generalized Support Software Reuse Asset Library » de la NASA.

En 1998, H. Lounis, H. A. Sahraoui et W. Melo publient l'article « Vers un modèle de prédiction de la qualité du logiciel pour les systèmes à objets » [28]. Le but de leurs travaux est de valider empiriquement l'hypothèse selon laquelle certaines formes de couplage sont préjudiciables à la qualité du logiciel, en utilisant une méthode statistique et une technique d'apprentissage symbolique. Par ailleurs, ils soutiennent que la modularité est un critère important de la qualité du logiciel.

En 1999, M. Almeida, Lounis H. et W. L. Melo [3] dans leur article « An Investigation on The Use of Machine Learned Models for Estimating Software Correctability » présente une étude empirique qui démontre que les algorithmes d'apprentissage automatique sont efficaces pour évaluer le coût de la maintenance corrective de composants logiciels défaillants.

Enfin, en 2001, dans l'article « Machine Learning Approach to Predict Software Evolvability using Fuzzy Binary Trees », M. Boukadoum, H. A. Sahraoui et H. Lounis [8] ont utilisé l'apprentissage automatique et la logique floue afin de prédire l'évolutivité d'un logiciel.

Ces travaux nous montrent l'importance des mesures dans le domaine du génie logiciel. Elles permettent d'estimer les coûts durant le cycle de vie du produit logiciel.

Les mesures d'attributs internes que nous venons de définir seront exploitées par les différents algorithmes d'apprentissage que nous allons présenter dans le chapitre suivant. Il s'agira de détecter les mesures pertinentes qui influent sur les attributs de qualité qui sont la maintenabilité, la réutilisabilité et la propension à engendrer des erreurs.

# Chapitre 3

# Les Algorithmes d'apprentissage

## 3.1 Introduction

Depuis longtemps l'homme a voulu créer des machines dotées d'intelligence. Néanmoins, c'est en 1956 lors d'une conférence à Dartmouth, aux États-Unis, qu'est utilisé pour la première fois le terme « Artificial Intelligence ». L'idée des scientifiques réunis, dont John McCarthy et Marvin Minsky, était d'aboutir à des programmes d'ordinateurs pourvus d'intelligence. En effet, en réalisant ces programmes capables de raisonner, on pouvait espérer résoudre des problèmes que l'homme ne peut résoudre.

L' apprentissage automatique (AA) est l'un des champs d'étude de l'intelligence artificielle. Nonobstant une puissance relativement faible des machines de calculs des années soixante et soixante-dix, les techniques d'apprentissage automatique ont connu un développement sans doute lent mais continu. L'objectif de l'AA est de construire des machines intelligentes capables d'apprendre automatiquement à partir de sa propre expérience. On essaie de reproduire certains aspects de l'intelligence humaine au moyen de différentes méthodes simulant les processus de raisonnement humain tels l'inférence, la

déduction, l'abduction, l'induction, ceci, en s'appuyant ou pas sur des connaissances.

L'AA a énormément progressé en termes de compréhension théorique mais également en applications réelles dans divers domaines. La raison principale est due à la croissance fulgurante des puissances de calcul. Cela rend les applications d'apprentissage possibles par des algorithmes d'apprentissage automatiques très exigeants en temps de calcul. Réciproquement, les algorithmes d'apprentissage permettent une meilleure utilisation de la puissance de calcul existante en exploitant systématiquement l'information contenue dans les bases de données. La disponibilité de bases de données de grandes envergure facilite l'exploitation des données et, par là même, la progression de la recherche dans l'apprentissage automatique.

L'AA englobe l'analyse statistique et la modélisation de données ainsi que les réseaux neurologiques artificiels et l'Apprentissage Symbolique Automatique (ASA).

Nous allons résumer les trois types de raisonnement cognitif humain mis en œuvre en apprentissage automatique [62]:

- L'apprentissage par analogie : lorsqu'on apprend par analogie, on essaie de transférer des connaissances d'une tâche bien connue vers une autre qui ne l'est pas. Ainsi, de nouvelles solutions sont susceptibles d'être dérivées à partir de concepts ou de solutions similaires connues. Le transfert et la similarité sont deux notions qui découlent immédiatement de l'apprentissage par analogie. Les systèmes de raisonnement par cas, ou par similarité, sont davantage des méthodes de raisonnement que des méthodes d'apprentissage en tant que telles. Ces dernières implémentent des fonctions de similarité sans utiliser de mécanisme d'adaptation.

• L'apprentissage par déduction : utilise des connaissances préexistantes à partir d'un raisonnement déductif qui augmente l'information fournie. Les méthodes de ce type d'apprentissage, « Analytical Learning », [49] utilisent les connaissances préexistantes pour analyser, ou expliquer, un exemple observé lors de l'apprentissage qui couronne de succès les concepts existants. Par la suite, cette explication permet de différencier les attributs pertinents de l'exemple d'apprentissage de ceux qui ne le sont pas. Ainsi, l'exemple sera généralisé par un raisonnement logique.

•L'apprentissage par induction : le but de cet apprentissage est d'obtenir des règles générales représentant les connaissances obtenues à partir d'exemples. Les règles ainsi obtenues peuvent être représentées d'une façon explicite (i.e. les règles sont facilement interprétables) ou d'une manière implicite avec un codage qui n'est pas toujours facile à interpréter. À partir d'un ensemble d'exemples, l'algorithme d'apprentissage doit produire des règles de classification permettant de classer les nouveaux exemples en suivant une méthode de généralisation de connaissances. Les approches les plus connues que nous pouvons évoquer sont les réseaux de neurones artificiels et les arbres de décisions.

Pour la mise au point de notre recherche, nous avons utilisé des algorithmes d'apprentissage de plusieurs types. Nous allons présenter ceux qui sont exploités au cours de nos travaux:

- Les algorithmes de la plate forme Weka [66]
- Induction de règle : CN2 [21]
- Classificateur oblique : OC1 [54]

## 3. 2 Les algorithmes de la plate forme Weka

Le système ***Weka*** (Waikato Environnement For Knowledge Analysis) est un système d'extraction de connaissance complet diffusé sous licence GNU. Il propose une collection d'algorithmes pour la résolution de problèmes de data mining. ***Weka*** a été développé à l'université de Waikato en Nouvelle Zélande. Il est écrit en java ce qui le rend très portable. Il fonctionne sur toutes les plate-forme (Unix, Windows et MacOS) et renferme une interface graphique d'utilisation facile.

Le code source de Weka est gratuit et il est aussi possible d'ajouter de nouvelles fonctionnalités à ce système. Le programme weka est téléchargeable à partir du site : http://www.cs.waikato.ac.nz/~ml/weka/index.html, et l'installation de ce logiciel est simple. Néanmoins, il est nécessaire d'avoir installé le JDK au préalable.
Les données manipulées par Weka peuvent être enregistrées dans une base de données ou dans un fichier. Dans le cas d'un fichier, il faut toutefois le convertir dans le format "ARFF". Notons que l'on peut travailler avec deux fichiers. Le premier contient les instances de l'ensemble d'apprentissage (Training) tandis que le second renferme l'ensemble des données de test.

Voici les différentes familles d'algorithmes disponibles dans Weka :

- Algorithmes de classification : Comme Naive Bayes, rule learner, décision tables, etc.
- Methodes de clustering : Comme Cobweb et Em algorithme.
- Algorithme de recherche associative : Le seul disponible est APRIORI.

Les algorithmes disponibles sont très nombreux. Nous avons testés les algorithmes de chacune de ces familles et nous avons retenu ceux qui nous ont

Les algorithmes disponibles sont très nombreux. Nous avons testés les algorithmes de chacune de ces familles et nous avons retenu ceux qui nous ont offerts des résultats satisfaisants. Nous allons donc présenter ceux qui ont été retenus :

### 3.2.1 L'algorithme M5'

M5' est un algorithme qui fonctionne exclusivement avec des valeurs numériques. Il est basé sur le modèle linéaire de régression, une technique à considérer lorsque la classe et les attributs sont numériques. M5' implémente trois modèles différents nommés:

- Régression linéaire.
- Arbres de régression.
- Modèles d'arbres.

L'idée de la régression linéaire est d'exprimer la classe sous forme linéaire en fonction des attributs, ceci en conférant un poids pour chaque attribut:

$C=w_0+w_1\ a_1+w_2\ a_2+w_3\ a_3+\ldots\ldots\ldots\ldots w_k a_k$

Où

C : est la classe

$a_1, a_2, a_3\ldots\ldots a_k$: valeurs des attributs

$w_0, w_1, w_2, w_3\ \ldots\ldots\ldots\ldots w_k$: Le poids des attributs.

Cette formule nous donne la prédiction de la classe de chaque instance. Il est à noter que les poids sont calculés à partir de l'ensemble d'apprentissage. L'objectif de la régression linéaire est de choisir le bon coefficient $W_j$ qui minimise la différence entre la valeur prédite de la classe et sa valeur actuelle. .

Le résultat de l'algorithme M5' est une expression linéaire qui représente chaque classe. Pour classer un exemple, on calcule la valeur de chaque expression linéaire en remplaçant les attributs par ceux de l'exemple puis, on prend la valeur de la classe qui est la plus grande.

Arbres de régression et modèles d'arbres sont tous les deux similaires du fait qu'ils génèrent des arbres de décision avec des sorties numériques au niveau des nœuds feuilles plutôt que des sorties catégoriques. Ces deux modèles sont toutefois différents au niveau de la nature exacte de ces sorties : les arbres de régression produisent une valeur numérique moyenne pour les sorties tandis que les modèles d'arbres produisent des équations linéaires au niveau de chaque nœud feuille.

Si on prend un exemple de règle généré par M5' :

Inline_comments>25.5 :

Executable>137.5 :**Lm1**

Cela signifie que si la règle est vraie (sans oublier que tous les nœuds parents doivent l'être aussi), la sortie maintenance qui est la variable à prédire est décidée par l'équation linéaire Lm1.

**Lm1**: maintenance=-0.0208* inline_comments +0.00019* blank_lines +1.8059

Pour évaluer cette équation, on doit simplement remplacer tous les attributs numériques (inline_comments et blank_lines dans cet exemple) avec leur valeur pour l'exemple particulier traité. Comme on l'a déjà cité, les modèles d'arbres et les arbres de régression diffèrent dans le format de l'équation de sortie. Pour l'exemple, dans le cas des modèles d'arbres, la règle 1 change et prend la forme d'une valeur numérique moyenne.

**Avantages et limitations de M5':**

- Les modèles d'arbres combinent les arbres de décision avec une régression linéaire statistique. Cette technique est appropriée quand les relations entre les attributs d'entrées et de sorties ne sont pas linéaires.
- M5' est limité aux prédictions des sorties numériques.
- Un manque d'explication concernant ce qui a été appris (dans la phase d'apprentissage) peut constituer un problème.

### 3.2.2 OneR (1R)

OneR constitue une approche pour trouver des règles de classification simples à partir d'un ensemble d'apprentissage. Ce programme génère un arbre de décision à un seul niveau qui est exprimé sous la forme d'un ensemble de règles avec pour chaque règle, un test sur un seul attribut.

Il arrive que ces règles simples donnent un taux d'exactitude plus élevé que les algorithmes d'apprentissage sophistiqués tels que J48 [66] ou CN2 [21]. Cela peut s'expliquer par le fait qu'un seul attribut est souvent suffisant pour classer un nouvel exemple.

L'idée avec OneR est d'élaborer des règles qui testent un seul attribut ; chaque branche émanant d'un attribut correspondant à une valeur différente de ce dernier. Il est évident qu'on associe à chaque branche la classe qui occure le plus souvent dans l'ensemble d'apprentissage. On détermine l'erreur des règles facilement, en comptabilisant le nombre d'erreurs qui surviennent. Ce nombre représente les exemples qui sont mal classés.

Chaque attribut génère un ensemble différent de règles : une règle pour chaque valeur de cet attribut. On évalue l'erreur pour chaque ensemble de règles associés à l'attribut.

Voici le pseudo code de cet algorithme :

Pour chaque attribut

Pour chaque valeur de cet attribut, élaborez une règle comme suit :

- Compter le nombre de fois où chaque classe apparaît ;
- Trouver la classe qui apparaît le plus fréquemment ;
- Associer cette classe à la règle assignée à cette valeur de l'attribut ;
- Calculez le taux d'erreurs de ces règles ;
- Choisir la règle qui a le plus petit taux d'erreurs.

### 3.2.3 Réseaux de neurones

Les réseaux de neurones (RN) ont été introduits pour la première fois en 1943 par J.Mc Culloch et W.Pitts [48] en reproduisant le fonctionnement d'une machine de Turing. Le premier modèle de réseau est apparu au début des années 50, mais c'est à la fin de celles-ci que Frank Rosenblatt a présenté le Perceptron, premier modèle démontrant un réel processus d'apprentissage. Néanmoins, c'est à partir des années 80 qu'on a pu assister à l'expansion véritable de ces méthodes grâce notamment à des auteurs comme Kohonen(1982), Jordan(1988) et Elman(1990) qui ont mis au point de nouvelles architectures de réseaux de neurones.

#### 3.2.3.1 Définition

Selon [29], un réseau de neurones est un processeur massivement distribué en parallèle qui a une propension naturelle pour stocker de la connaissance empirique et la rendre disponible à l'usager. Il ressemble au cerveau sur deux aspects :

1. La connaissance est acquise par le réseau au travers un processus d'apprentissage.
2. Les connexions entre les neurones, connues sous le nom de poids synaptiques, servent à stocker la connaissance.

Nous pouvons ainsi définir l'architecture d'un réseau de neurones comme étant un ensemble de petites unités de calculs (les neurones) reliées entre elles par des liens de communication (les connexions). Ce qui leur permet de disposer d'un canal pour envoyer et recevoir des signaux en provenance d'autres cellules du réseau. Chacune de ces connexions reçoit une valeur et un poids, une pondération en quelque sorte. Les cellules disposent d'une entrée qui facilite la réception de l'information des autres cellules
L'information transportée par ces connexions est de type numérique. Chaque unité réalise un calcul à partir de données issues de ces connexions et de données locales [62].

Les RN sont utilisés pour leur capacité à apprendre à partir d'exemples bruités comme les caméras ou les micros (reconnaissance de forme ou de son). Ils sont appropriés lorsque le temps d'apprentissage n'est pas essentiel. Ce temps est en effet souvent très supérieur à celui d'autres méthodes comme les arbres de décision. Par contre, la classification d'un nouveau cas est très rapide. Dans le paragraphe suivant, nous allons définir avec plus de détails les neurones.

### 3.2.3.2 Neurone Formel

Le neurone artificiel est une unité de calcul combinant des entrées réelles $x_1,\ldots,x_n$ en une sortie réelle o. Lorsque les neurones sont organisés selon plusieurs couches, les sorties d'une couche alimentent les entrées de la prochaine. Dans ce cas, les entrées $X_i$ peuvent correspondre à des unités dont

chacune possède un état d'activation qu'elle propage aux autres unités au moyen de poids synaptiques $W_i$ :

- Si $W_i>0$, alors l'état d'activation correspond à une synapse excitatrice.
- Sinon l'état d'activation correspond à une synapse inhibitrice.

Du point de vue biologique, un neurone émet un signal en fonction des signaux qui lui proviennent des autres neurones au moyen de sommations des signaux. Dans un neurone artificiel, un opérateur de sommation permet de simuler le potentiel post synaptique et correspond à la somme de chaque poids $W_i$ avec chaque entrées $X_i$.

Les poids synaptiques des unités du réseau changent le comportement d'activation du réseau ce qui lui permet d'apprendre un nouveau comportement. Ainsi, le réseau est capable d'établir des associations entre entrées et sorties. La méthode utilisée pour modifier le comportement d un réseau est nommée règle d'apprentissage.

Dans la figure suivante (Voir FIG. 3.1) nous allons présenter la version moderne du modèle développé par Mc Culloch et Pitts.

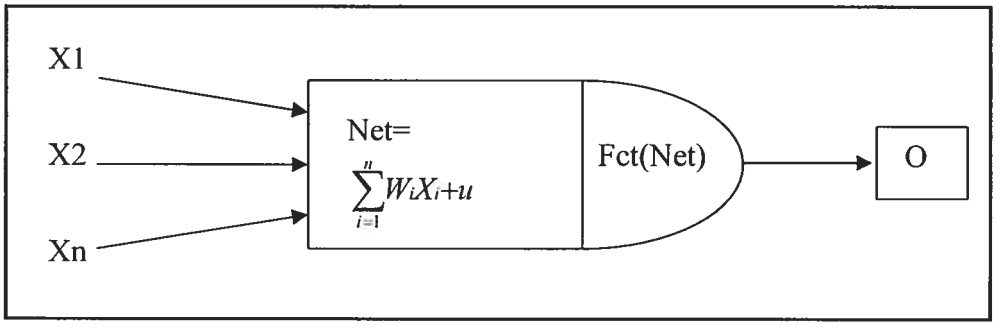


**FIG. 3.1: Neurone Formel**

Entrées $X_i$ :

Les valeurs d'entrée sont des stimuli externes ou les sorties d'autres neurones artificiels. Elles peuvent être discrétisées ou réelles.

Poids $W_i$ :

Le neurone artificiel calcule la somme pondérée de ses entrées. Les poids sont des valeurs réelles qui déterminent la contribution de chaque entrée. Le but d'un algorithme d'apprentissage pour un neurone est de déterminer le meilleur ensemble de poids pour le problème en cours.

Biais/Seuil $X_0$

Le biais est un nombre réel qui est additionné à la somme pondérée des valeurs d'entrée. Le seuil est aussi un nombre réel mais est soustrait à la somme pondérée des valeurs d'entrée.

Pour simplifier, le biais peut être vu comme une entrée supplémentaire fixe associée à un poids, où $W_0$=u et $X_0$=+1. Dans le cas du seuil on a $X_0$=-1.

Fonction d'activation :

La première fonction d'activation de McCulloch et Pitts était la fonction seuil. Actuellement, nous retrouvons un grand nombre de fonctions d'activation comme la sigmoïde, la courbe linéaire par parties et la gaussienne. Signalons que la fonction d'activation de type sigmoïde est souvent utilisée dans le cas de la classification.

Il existe plusieurs types d'architecture de réseaux de neurones. La famille la plus répandue des RN est celle des réseaux non-récurrents comme le perceptron multicouches. Dans ce type d'architecture, les neurones sont uniquement reliés au moyen de connexion inter-couches. Les réseaux récurrents utilisent

l'ensemble des connexions possibles qui permettent de prendre en compte l'état du réseau. Il s'agit en l'occurrence de réseaux avec mémoire. Il est important de noter que des architectures de RN nécessitent des types différents d'algorithmes d'apprentissage.

L'outil de datamining WEKA intègre le modèle de perceptron multicouches qui fait partie des réseaux non-réccurents.

### 3.2.3.3 Le perceptron multicouches

Le perceptron multicouches est un réseau comportant L couches. Chaque neurone d'une couche étant totalement connecté aux neurones de la couche suivante. L'architecture du réseau peut être décrite par le nombre de couches et le nombre de neurones dans chaque couche. De même que pour les neurones, dans une architecture multicouches de RN il y a [62]:

- **Les nœuds d'entrée** : c'est la première couche (couche d'entrée). Elle sert à recevoir les données source que l'on veut analyser.
- **Les nœuds cachés** : c'est la seconde couche (couche cachée). Elle n'a qu'une utilité intrinsèque pour le réseau et n'a pas de contact direct avec l'extérieur.
- **Les nœuds de sortie** : c'est la troisième couche (couche de sortie). Elle donne le résultat obtenu après compilation par le réseau des données d'entrée.

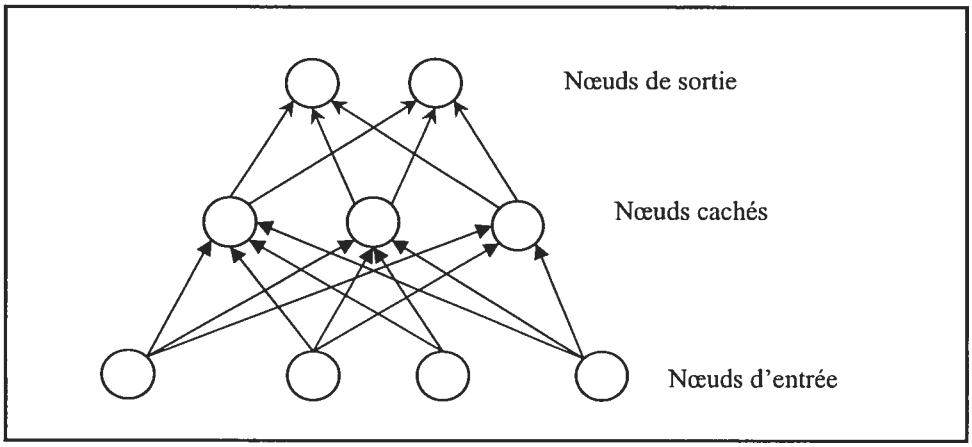


**FIG. 3.2: Réseau multicouches**

Dans les réseaux de neurones, il existe deux types principaux de connexions : les connexions Feed-Forward où l'information est transportée d'un neurone vers un autre dans le sens de flux du réseau et les connexions Feed-Back qui est une connexion retour : on renvoie au neurone la réaction du neurone récepteur.

L'apprentissage d'un réseau de neurones revient à la détermination des poids du réseau grâce aux algorithmes de calcul. Il consiste tout d'abord à calculer les pondérations optimales des différentes liaisons en utilisant un échantillon. La méthode la plus utilisée est la *backpropagation* : on entre des valeurs dans les cellules d'entrée et en fonction de l'erreur obtenue en sortie, on corrige les poids accordés aux pondérations. C'est un cycle qui est répété jusqu' à ce que la courbe d'erreurs du réseau ne soit plus croissante . Il en est de même pour d'autres méthodes d'apprentissage telles que le *quickprop* par exemple. Mais la plus utilisée reste encore la rétro-propagation.

L'algorithme de rétropropagation consiste à effectuer une descente du gradient sur la fonction de l'erreur quadratique E, cet algorithme d'apprentissage est dérive des formules qui suivent :

Erreur quadratique : $E = \frac{1}{2}\sum_i (D_i - A_i)^2$

Adaptation des poids synaptiques par descente du gradient : $\Delta W_{ij} = -\alpha \frac{\partial E}{\partial W_{ij}}$

Calcul du gradient de l'erreur par rapport au poids synaptique :

Unité de sortie avec une sortie linéaire (Ai = Si)

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial A_i}\frac{\partial A_i}{\partial W_{ij}} = \partial_i X_j$$

Di : Sortie désirée de l'unité i, provenant de la base d'apprentissage.

Ai : Valeur obtenue à la sortie de l'unité 'i' après activation.

$\Delta W_{ij}$ : Valeur du changement effectué sur le poids Wij.

E : Mesure de l'erreur (fonction d'erreur quadratique).

Xj : Valeur de l'entrée 'j' de l'unité en question.

Wij : Poids synaptique associé à l'entrée 'j' de l'unité 'i'

$\alpha$ : Valeur constante qui sert à contrôler la vitesse de convergence de l'apprentissage (pas d'apprentissage - *learning rate*). Cette valeur est comprise dans l'intervalle [0..1].

L'algorithme effectue ainsi une descente de gradient de la surface d'erreur dans l'espace des poids. Cet algorithme se caractérise par sa lenteur d'exécution. Des propositions d'améliorations ont déjà été entamées.

### 3.2.4 IBK (Instance Based K-Neighbors)

IBK est l'implémentation de Knn (K-nearest neighbor) dans weka [66]. IBK diffère des traditionnelles méthodes d'apprentissage car aucun modèle n'est

induit à partir des exemples. Les données ne subissent aucune modification. Elles sont simplement sauvegardées en mémoire.
Il s'agit de faire de l'apprentissage à partir de cas [64]. Les instances les plus proches sont classifiées ; on leur attribue une classe. Afin de classifier un nouveau cas, l'algorithme cherche les K plus proches voisins de ce nouveau cas et prédit la réponse la plus fréquente de ces K plus proches voisins.
Deux paramètres sont pris en compte: le nombre K et la fonction de similarité pour établir les comparaisons avec des cas déjà classés.

En général le choix de ces deux paramètres est arbitraire mais très important car les résultats de la classification en dépendent. Cet algorithme s'applique très bien dans le cas où les attributs sont numériques car le regroupement des instances se fait sur la base d'une fonction qui calcule une distance entre attributs. En général la fonction utilisée est la distance euclidienne.
Les performances de la méthode dépendent du choix de la distance, du nombre de voisins et du mode de combinaison des réponses des voisins. En générale, les distances simples fonctionnent correctement. Dans le cas contraire, il faut envisager le changement de distance ou l'adoption d'une autre méthode.
Knn permet de traiter des problèmes avec un grand nombre d'attributs. Dans ce cas, nous devons disposer d'un grand nombre d'exemples. En effet, pour que la notion de proximité soit pertinente, il faut que les exemples couvrent bien l'espace et soient suffisamment proches les uns des autres. Si le nombre d'attributs pertinents est faible relativement au nombre total d'attributs, on obtient de mauvais résultats car la proximité sur les attributs pertinents sera noyée par les distances sur les attributs non pertinents. Nous devons donc procéder à une sélection des attributs pertinents.
Prenons l'exemple suivant :

Supposons que D1 est un nouveau document à classer. Si on prend k=1 alors D1 sera classé +. Par contre si, on prend k=5, le même document sera classé – car les 3 plus proches voisins portent le signe – comme on le voit bien dans la figure 4.3 qui est connue sous le nom de diagramme de Voronoi.

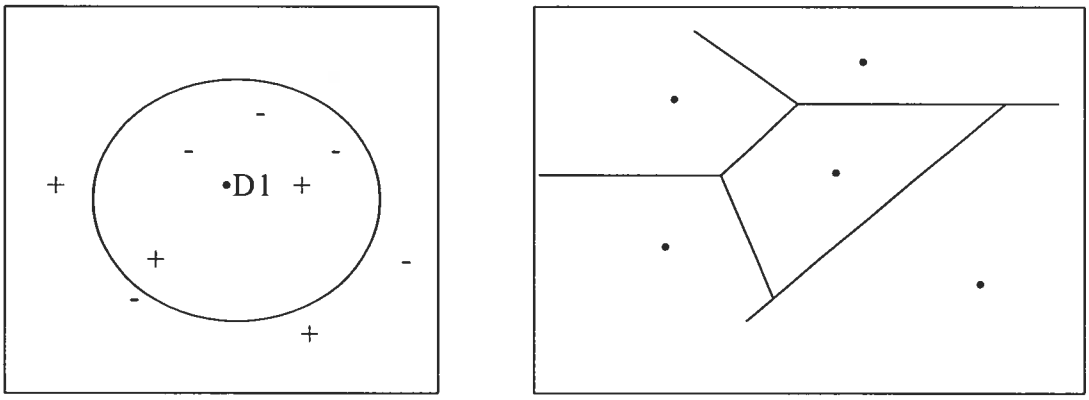


**FIG. 3.3: Diagramme de Voronoi**

**Avantages IBK**

- le temps d'apprentissage inexistant (car les données sont sauvegardées telles quelles).
- les expériences ont prouvé que KNN résiste bien aux données bruitées.
- Clarté des résultats : bien que la méthode ne produit pas de règle explicite, la classe attribuée à un exemple peut être expliquée en exhibant les plus proches voisins qui ont amené à ce choix.

**Inconvénients IBK**

- le temps de calcul est important :Avec les arbres de décision on teste un seul attribut à chaque nœud de l'arbre, tandis que KNN utilise tous les

attributs d'un cas pour calculer la similarité avec un nouveau cas à classer.

- un espace mémoire important est requis afin de sauvegarder les données ainsi que des méthodes d'accès rapides pour accélérer les calculs.

### 3.2.5 Machines à support de vecteurs (ou SVM)

SVM est l'acronyme de Support Vector Machines. Cette technique et ce terme ont été inventés principalement par Vapnik [65]. Certains auteurs l'appelle aussi Séparateurs à Vaste Marge.

Avec ce genre de technique on parlera fréquemment de la classe + et la classe – ainsi que d'exemples négatifs et d'exemples positifs [22]. Le but de SVM est de déterminer un hyperplan permettant de séparer les exemples positifs et les exemples négatifs le mieux possible. L'hyperplan qui sépare efficacement ces deux ensembles est celui où la distance entre l'hyperplan et les exemples positifs et négatifs les plus proches soit la plus grande possible.
On cherche donc $h$ sous forme d'une fonction linéaire : $h(\boldsymbol{x}) = w.\boldsymbol{x} + b$
La surface de séparation est donc l'hyperplan :
$w.x+b=0$ qui est valide si $u_i.h(x_i) \geq 0$

En supposant qu'il existe un hyperplan permettant de séparer les exemples positifs des exemples négatifs, nous devons chercher celui qui passe « au milieu » des points des deux classes d'exemples. Cela revient à chercher l'hyperplan le « plus sûr ». En effet, supposons qu'un exemple n'ait pas été décrit parfaitement, une petite variation ne modifiera pas sa classification si sa distance à l'hyperplan est grande.

Formellement, cela revient à chercher un hyperplan dont la distance minimale aux exemples d'apprentissage est maximale (Voir FIG. 3.4). Les exemples les plus proches qui suffisent à déterminer cet hyperplan sont appelés vecteurs de support ou encore exemples critiques. La distance séparant l'hyperplan de ces points est appelée « marge ».Du fait qu'on cherche à maximiser cette marge, dès lors, on parlera de *méthode des séparateurs à vaste marge.*

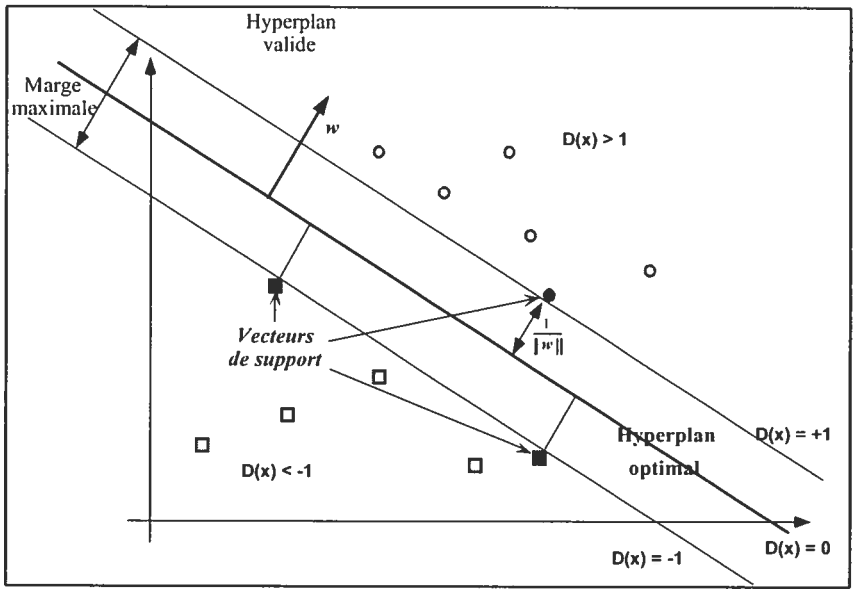


**Fig 3.4 Hyperplan avec marge optimale**

La distance d'un point à l'hyperplan est :

$$d(\boldsymbol{x}) = \frac{|\boldsymbol{w}.\boldsymbol{x} + w_0|}{\|\boldsymbol{w}\|}$$

L'hyperplan optimal est celui pour lequel la distance aux points les plus proches (***marge***) est maximale. Cette distance vaut :

$$\frac{2}{\|\boldsymbol{w}\|}$$

Maximiser la marge revient donc à minimiser $\|w\|$ sous les contraintes suivantes :

$$\begin{cases} \min \frac{1}{2}\|w\|^2 \\ \forall i \quad u_i(w.x_i+w_0) \geq 1 \end{cases}$$

Une fois un tel hyperplan déterminé pour chaque classe, il peut être utilisé pour savoir si un nouvel exemple peut être classé dans cette classe ou pas.
L'efficacité des SVM est supérieure à celle de toutes les autres méthodes sur la classification de textes. Son efficacité est aussi avérée pour la reconnaissance de formes. Un autre intérêt est la sélection de Vecteurs Supports qui représentent les vecteurs discriminants grâce auxquels est déterminé l'hyperplan. Les exemples utilisés lors de la recherche de l'hyperplan ne sont alors plus utiles et seuls ces vecteurs supports sont utilisés pour classer un nouveau cas. Cela en fait une méthode très rapide.

### 3.2.6 Générateurs de règles (PART)

PART est un générateur de règles qui utilise le J48 (décrit dans 3.2.7) pour générer un arbre de décision à partir duquel il extrait ses règles [66]. PART est applicable sur des attributs d'entrées qui sont numériques ou catégoriques (l'attribut à prédire ne doit pas être numérique).

Dans [11], Frank et Witten décrivent une expérience mise au points sur plusieurs ensembles de données en adoptant les algorithmes C4.5 [58] et PART [66]. Le résultat de cette expérience démontre que PART surpasse largement l'algorithme C4.5.

L'avantage principal de PART par rapport à C4.5, est que contrairement à C4.5, l'algorithme de génération de règles n'a pas besoin de faire une optimisation pour produire un ensemble précis de règle.
Afin de produire une règle unique, on doit construire un arbre de décision élagué. Dans ce cas, on prend seulement les feuilles qui couvrent un large nombre d'exemples et on les établit sous forme d'une règle.

Il y'a deux paramètres qui affectent les performances de l'algorithme PART :
• le facteur de confiance
• le nombre minimum d'exemples par feuille.

Le premier paramètre détermine la valeur de confiance à utiliser lorsque l'arbre de décision est élagué. Cela se traduit par l'élimination des branches qui fournissent un faible gain dans l'exactitude statistique du modèle ou celles qui ne fournissent aucun gain.

Généralement, la valeur de 25% donnée au facteur de confiance fournit de bons résultats. Dans le cas où le taux d'erreur de l'ensemble de test est supérieur au taux d'erreurs de l'ensemble d'apprentissage, il faut diminuer le facteur de confiance pour obtenir un modèle de données plus large.

Si on augmente le facteur de confiance, l'élagage de l'arbre de décision va être diminué. Dans ce cas de figure, on obtient un modèle de données plus spécifique qui se base sur l'ensemble d'apprentissage.

Le nombre minimum d'exemples par feuille : détermine le nombre minimum d'instances que l'on doit avoir dans chaque nouvelle feuille crée dans l'arbre de décision. Ce paramètre permet aussi de concevoir un arbre de décision généralisé ou spécialisé. Une valeur élevée permet de créer un arbre généralisé et une petite valeur crée un arbre plus spécialisé.

**Avantage et limitation**

- Les règles de PART sont plus facile à interpréter que les structures d'arbres, surtout dans le cas où on utilise des attributs numériques (attributs d'entrées).
- Le processus de simplification de règles peut donner des exemples mal classés.
- Les règles de PART ne permettent pas la prédiction des attributs numériques (attributs de sortie)

### 3.2.7 C4.5 (J48)

C4.5 [58], [66] fait partie des méthodes d'apprentissage fondées sur les arbres de décision (voir annexe c). C'est un descendant amélioré de l'algorithme ID3 [37], [38].
Il permet de prendre en compte des ensembles d'apprentissage renfermant des valeurs inconnues et des attributs de types continus. Il offre aussi la possibilité de procéder à un élagage de l'arbre de décision induit et d'en dériver des règles de production.
Ces améliorations permettent ainsi de considérer un éventail d'application possible pour C4.5 beaucoup plus vaste que le champs de son ancêtre ID3.

Notons que la version de C4.5 intégrée dans la plate forme d'outils pour l'apprentissage weka se nomme J48.

C4.5 fait partie de la méthode « Diviser pour Régner »; c'est à dire diviser récursivement et le plus efficacement possible les exemples de l'ensemble d'apprentissage par des tests définis à l'aide des attributs jusqu'à l'obtention des

sous-ensembles d'exemples ne contenant (presque) que des exemples appartenant tous à une même classe. Le principe de fonctionnement de l'algorithme peut être résumé de la façon suivante :

Si tous les exemples appartiennent à la même classe

Alors

- Créer une feuille qui porte le nom de la classe;

Sinon

- Sélectionner un test basé sur un attribut (le meilleur).
- Subdiviser l'ensemble d'apprentissage en sous-ensembles. chacun d'eux associés à une valeur possible de l'attribut teste. Réappliquer la même procédure à chacun des sous-ensembles ainsi générés.

Fin_si

L'étape clé de ce type d'algorithme est le choix du « meilleur » attribut à tester afin d'obtenir des arbres de décisions compacts et possédant la meilleure capacité prédictive. À cet égard, l'algorithme calcule des arbres de décisions dont l'erreur apparente est faible. En effet, l'algorithme procède de façon descendante sans jamais remettre en question les choix effectués auparavant. Il ne peut jamais exclure qu'un autre choix de test conduise à un meilleur arbre. Les problèmes qui découlent de cette approche, sont que l'erreur apparente est une vision très optimiste de l'erreur réelle, et que de trouver un arbre de décision d'erreur apparente minimale est, en général, un problème NP-complet. Pour l'algorithme ID3 Le choix du test se fait par l'utilisation des fonctions d'entropie et de gain :

$$\text{Entropie(p)} = -\sum_{i=1} P(\frac{i}{p}) \ln(P(\frac{i}{p}))$$

$$\text{Gain}(p,t) = \text{Entropie}(p) - \sum_{i=1}^{n} P_i * Entropie(P_i)$$

Avec p un patron de l'ensemble d'apprentissage.

$P(\frac{i}{p})$ la probabilité d'obtenir au hasard un patron de classe i de l'ensemble d'apprentissage,

T un test d'arité n,

$P_i$ est la proportion d'éléments de l'ensemble des exemples associés à p qui vont sur le noeud en position pi (i ème fils du noeud en position p).

Dans l'ensemble des tests disponibles, chaque étape ne peut envisager que des tests pour lesquels il existe au moins deux branches de deux éléments au moins (valeur par défaut). Si aucun test ne satisfait à cette condition, le nœud est terminal. De ce fait, tous les exemples appartiennent à la même classe. La fonction Gain privilégie les attributs ayant un grand nombre de valeurs. Elle est donc pondérée par une fonction pénalisant les test qui répartissent les éléments en un trop grand nombre de sous-classes. Cette mesure de la répartition est appelée, par [60], « Split Information » et définie par :

$$\text{Split}(p,t) = -\sum_{i=1}^{n} P(\frac{i}{p}) \ln(P(\frac{i}{p}))$$

Il faut remarquer que, contrairement à la fonction entropie, cette définition est indépendante de la répartition des exemples à l'intérieur des différentes classes. La valeur de « Split information » ne dépend que de la répartition entre les différentes valeurs possibles pour le test. Cette fonction prend de grandes valeurs lorsque le test a un grand nombre de valeurs possibles avec peu d'éléments pour chacune des valeurs.

C4.5 utilise l'ensemble d'apprentissage pour élaguer l'arbre obtenu. Le critère d'élagage est basé sur une heuristique permettant d'estimer l'erreur réelle sur un sous-arbre donné.

Comme dans le cas de l'algorithme d'apprentissage Part, il y a deux paramètres qui affectent les performances de C4.5 :

• le facteur de confiance

• le nombre minimum d'exemples par feuille.

### 3.2.8 Naive Bayes

Nommées d'après le théorème de Bayes, ces méthodes sont qualifiées de "Naïve" ou "Simple" car elles supposent l'indépendance des variables. L'idée est d'estimer les probabilités à posteriori de toutes les classes. Pour chaque donnée, la classe avec la probabilité à posteriori la plus élevée est choisie comme prédiction.

La méthode naïve de Bayes suppose que les probabilités de certains événements peuvent être estimées par leurs fréquences et établit une hypothèse forte d'indépendance des attributs.

L'avantage de l'algorithme Naïf de Bayes par rapport à d'autres algorithmes est le fait que ce dernier ne requiert qu'une seule passe à travers l'ensemble d'apprentissage afin de générer le modèle de classification. Donc cette méthode peut obtenir des résultats plus satisfaisants que certains algorithme très sophistiqués comme le CN2 [21].

Cependant, il peut aussi donner de mauvais résultats sur un ensemble de données qui sont traités comme étant indépendantes alors qu'en réalité elles sont corrélées.

La formule de Bayes est une relation simple entre des probabilités.

Soit $D$ un langage de description, soit $\{1,...,c\}$ l'ensemble des classes, sous les hypothèses usuelles d'existence de lois de probabilité, la règle de classification de Bayes est la procédure qui, à toute description $d$ de $D$ associe :

$C_{Bayes}(d) = k \hat{I} \{1,...,c\}$ argmax $P(k/d) = k \hat{I} \{1,...,c\}$ argmax $P(d/k) \times P(k)$.

où $k$ argmax $f(k)$ retourne la valeur de $k$ qui maximise $f$. Mais, en règle générale, les quantités $P(d/k)$ et $P(k)$ ne sont pas connues. On suppose que $D$ est un produit cartésien de domaines, on suppose également disposer d'un échantillon $S$ d'exemples $(\vec{x},c(\vec{x}))$. On souhaite classer un élément $\vec{d}=( d_1,...,d_n)$. La règle de classification de Bayes s'écrit :

$$C_{Bayes}(\vec{d}) = k \hat{I} \{1,...,c\} \text{ argmax } P((d_1,...,d_n)/k) \times P(k)$$

Pour rendre la méthode effective, on souhaite remplacer $P((d_1,...,d_n)/k)$ et $P(k)$ par des estimations faites sur l'échantillon $S$. Pour toute classe $k$, on estime $P(k)$ par $\hat{P}(k)$ qui est la proportion d'éléments de classe $k$ dans $S$. Par contre, l'estimation des $P((d_1,...,d_n)/k)$ est difficile car le nombre de descriptions possibles peut être très grand et il faudrait un échantillon $S$ de taille trop importante pour pouvoir estimer convenablement ces quantités. On émet donc l'hypothèse simplificatrice suivante : *les valeurs des attributs sont indépendants connaissant la classe*. Cette hypothèse permet d'utiliser l'égalité suivante :

$$P((d_1,...,d_n)/k) = \prod_{i \in \{1,...,n\}} P(d_i/k)$$

Il suffit d'estimer, pour tout $i$ et toute classe $k$, $P(d_i/k)$ par $\hat{P}(d_i/k)$ qui est la proportion d'éléments de classe $k$ ayant la valeur $d_i$ pour le $i$ème attribut. Finalement, le classificateur naïf de Bayes associe à toute description $d$ la classe

$$C_{Bayes}(\vec{d}) = k \in \{1,...,c\} \text{ argmax } \prod_{i\in\{1,...,n\}} \hat{P}(d_i/k) \times \hat{P}(k)$$

Expression dans laquelle les probabilités sont estimées sur l'échantillon $S$. Cette méthode est simple à mettre en oeuvre. Bien qu'elle soit basée sur une hypothèse fausse en général (les attributs sont rarement indépendants), elle donne cependant de bons résultats dans les problèmes réels. Elle fournit un seuil de performance pour d'autres méthodes.

**Réseaux Bayésien**

Une variante des méthodes naive de Bayes sont les réseaux Bayésiens(RB) : dans ce modèle, on ne suppose plus que les variables soient toutes indépendantes. On permet à certaines d'être liées. Cela alourdit considérablement les calculs et les résultats n'augmentent pas de façon significative.

Un réseau Bayésien est une technique qui permet la représentation de la connaissance par un graphe clair qui intègre l'incertitude dans le raisonnement. Il présente aussi l'avantage de pouvoir prendre la décision lui-même en fonction d'un seuil de probabilité fixé.

L'idée fondatrice des réseaux bayésiens est d'exploiter les indépendances conditionnelles entre les variables afin de représenter la distribution des probabilités jointes.

Ainsi, on considère toutes les $P(X_1, ..., X_n)$ possibles.

Au lieu d'un nombre de valeurs exponentielles par rapport au nombre de variables, on a besoin, pour chaque variable, d'un nombre de valeurs exponentielles par rapport au nombre de ses parents.

Initialement, on part d'un réseau complètement connecté en ne gardant que les chemins qui lient des variables indépendantes.

- Si A est indépendante de B, il n'y a pas de chemin de A vers B.
- Si A est dépendante de B, mais P(A|B) = P(A|B, C) alors cette dépendance passe par C, et on garde le chemin A-C-B et on élimine le chemin A-B.

On recherche toutes les indépendances conditionnelles qui peuvent expliquer toutes les chaînes possibles.

Voici l' algorithme 'classique' de construction de RB :

Construire un graphe non orienté complet, G, sur l'ensemble de nœuds

$n = 0$

Répéter

Répéter

Sélectionner une paire ordonnée de nœuds adjacents $X$, $Y$, telle que $X$ a au moins $n$+1 nœuds adjacents, et un ensemble S de $n$ nœuds adjacents à $X$ autres que $Y$ Si $X$ et $Y$ sont indépendants conditionnellement à S, supprimer l'arc $X$—$Y$ et mémoriser S comme l'ensemble des séparateurs des nœuds $X$ et $Y$

Tant qu'il reste des nœuds $X$, $Y$ éligibles et des ensembles S non encore testés

$n = n$+1

Tant qu'il reste des nœuds adjacents $X$, $Y$ tels que $X$ a plus de $n$+1 nœuds adjacents

Pour chaque triplet $X$, $Y$, $Z$ dans la configuration $X$—$Y$—Z et tel que $X$ et $Z$ ne sont pas adjacents

Orienter les arcs dans le sens $X \rightarrow Y \leftarrow Z$ ssi $Y$ ne fait pas partie de l'ensemble de séparateurs de $X$ et $Z$.

Appliquer les règles d'orientation suivantes :

Si un arc non orienté se trouve dans la configuration $X \rightarrow Y — Z$, telle que $X$ et $Z$ ne sont pas adjacents, oriente l'arc dans le sens $X \rightarrow Y \rightarrow Z$

Si un arc non orienté relie de nœuds $X$, $Y$ qui sont également connectés par un chemin orienté de $X$ à $Y$, oriente $X — Y$ dans le sens $X \rightarrow Y$ jusqu'à ce qu'aucune règle ne soit plus applicable.

Dans ce qui suit nous allons présenter CN2 et OC1, deux algorithmes d'apprentissage qui ne font pas partie du système d'analyse weka.

## 3.3 Induction de règles : CN2

Le système d'induction de règles **CN2** a été développé initialement par Clark et Niblett (1987, 1989) et modifié plus tard par Clark et Boswell [20] .

L'induction de règles est une méthode dite de « couverture » représentant la connaissance induite sous la forme d'une expression logique disjonctive définissant chaque classe. AQR [59] de Michalski et Larson et son descendant CN2 appartiennent à cette famille d'algorithmes dont est résumé ci-dessous le principe général :

1. Trouver une expression conjonctive satisfaite pour certains exemples de la classe ciblée et aucunement par un exemple des autres classes.
2. Ajouter cette conjonction comme un élément de la disjonction recherchée.

3. Ne plus considérer les exemples satisfaisant la conjonction trouvée. Si des exemples de la classe ciblée subsistent, répéter le processus.

En général, ce type d'algorithme œuvre à produire une définition cohérente et complète de la classe ciblée. Autrement dit, une définition ne reconnaissant que les exemples de la dite classe (cohérence) et à laquelle tous les exemples adhèrent (complétude).

La différence majeure entre AQR [59] et CN2 [21] est que le premier renvoie une règle, c'est à dire une disjonction de conjonctions de sélecteurs, tandis que le second renvoie une liste ordonnée de règles à la condition que chacune soit une conjonction de sélecteurs. Dans ce cas, l'application des règles doit s'effectuer dans l'ordre avec lequel l'algorithme les fournit.
Pour parvenir à une liste ordonnée de règles, CN2 utilise deux critères :un critère de signifiance statistique et un critère de qualité inspiré directement du calcul de l'entropie de l'algorithme ID3 [37], [38] :

$$\text{Critère de signifiance : } 2^* \sum_{i=1} f_i \log(\frac{f_i}{p_i})$$

Avec $f_i$ la distribution de fréquence des exemples observés parmi les classes.

$$\text{Critère de qualité : Gain(p,t)= Entropie(p) - } \sum_{i=1}^{n} P_i^* Entropie(p_i)$$

L'ancienne version de CN2 se basait sur l'utilisation de la fonction d'entropie dans sa recherche et ne pouvait générer qu'une liste de règles ordonnées. Certaines améliorations ont été apporte à CN2 [20] nous citons :

- utilisation de Laplace pour l'estimation des erreurs
- génération de règles non ordonnées

### 3.3.1 Utilisation de Laplace

Dans le calcul du critère de qualité, l'ancienne version de CN2 se basait sur l'utilisation de l'entropie pour l'estimation des erreurs. Le problème avec cette métrique réside dans la tendance à sélectionner des règles spécifiques qui couvrent un nombre restreint d'exemples. Plus la probabilité de trouver des règles avec une grande exactitude augmente, plus les règles deviennent spécifiques. Dans le cas extrême, on peut tomber sur des règles spécifiques qui couvrent un seul exemple de l'ensemble d'apprentissage. Ce genre de règles qui couvrent un petit ensemble d'exemples sont indésirables particulièrement dans le cas où on est en butte à du bruit dans le domaine. L'exactitude de ces règles sur l'ensemble d'apprentissage ne reflète pas leur véritable exactitude prédictive sur l'ensemble de test. Pour palier à ce problème, la fonction de Laplace a été ajoutée au niveau de l'algorithme CN2. Sa formule s'énonce comme suit :

$Laplaceaccuracy=(n_c+1)(n_{tot}+k)$

Où k est le nombre de classes dans le domaine.

$n_c$ le nombre d'exemples dans la classe prédite couverts par la règle.

$n_{tot}$ est le nombre total d'exemples couverts par la règle.

Quand on génère une liste de règles, la classe c prédite par une règle est tout simplement la classe qui arbore le plus d'exemples couvert par cette règle; ce qui résoud le problème de règle spécifique qui ne couvre qu'un seul exemple.

Plusieurs expériences ont été mises au point pour mesurer l'amélioration de l'exactitude prédictive en utilisant la fonction de Laplace. Celles-ci ont démontré une grande amélioration des résultats (Voir annexe B).

### 3.3.2 Génération de règles non ordonnées

L'algorithme original de CN2 génère des règles assemblées dans un ordre particulier.

Durant la classification d'un nouveau cas, chaque règle est testée dans l'ordre jusqu'à ce qu'une de ces règles convienne à ce cas. Par la suite, l'algorithme de classification s'arrête en assignant la classe que la règle a prédit pour cet exemple. Le problème avec les règles ordonnées est due au fait que la signification de chaque règle unique dépend de toutes les règles qui la précèdent dans la liste des règles [20]. Considérons l'exemple suivant :

Si plumes=oui alors classe= oiseau

Sinon si pieds=deux alors classe=humain

Sinon…….

La règle « si pieds=deux alors classe=humain », prise à part, est incorrecte car même un oiseau possède deux pieds. C'est pour cette raison que pour comprendre une règle il faut prendre en considération toutes les règles précédentes dans la liste. Ce problème devient plus complexe si on est confronté à un grand nombre de règles. La difficulté s'accroît pour un expert de saisir le véritable sens d'une règle qui se trouve dans le bas de la liste. Et comme les règles doivent généralement être validées par un expert avant de les utiliser dans une application , dès lors, l'utilisation de ce genre de règle ordonnée constitue un inconvénient contraignant. Pour pallier à ce problème, la nouvelle version de CN2 permet de générer une liste de règles non ordonnées. Dans cette mouture, toutes les règles peuvent être interprétées

individuellement, séparées l'une de l'autre. La modification principale qui a été apportée à l'algorithme CN2 consiste à réitérer la recherche pour chaque classe alternativement, en enlevant seulement les exemples couverts par cette classe lorsqu'une règle a été trouvée.

## 3.4. Classificateur oblique : OC1

OC1 (Oblique Classifier) est une application qui permet de construire des arbres de décision obliques à partir d'exemples. Les arbres de décision obliques sont des arbres sur lesquels chaque nœud peut contenir un test multi-variables (linéaires) dans les attributs des données. OC1 construit également des arbres qui contiennent des tests axés sur un attribut pour chaque nœud. Pour ce faire, OC1 partitionne l'espace des exemples au moyen d'hyperplans d'axes parallèles et obliques. Les arbres de décision obliques sont une extension des arbres de type axes parallèles. Cependant, l'utilisation des tests multi-variables qui en résultent peuvent s'avérer plus petits et/ou plus précis. Néanmoins, lors de l'induction, ils peuvent prendre plus de temps, beaucoup plus que les arbres uni-variables.

OC1 essaie de diviser l'espace dimensionnel des attributs en régions homogènes, des régions qui contiennent des exemples de même catégorie. Le but est de fractionner l'espace afin de réduire au minimum l'impureté de l'ensemble d'apprentissage. L'implémentation d'OC1 inclus six mesures : le gain d'information, le critère de Gini, la règle de pairage, la minorité maximale et celle de la somme ainsi que la somme de variances [54].

La fonction de Gini initiale est la suivante :

$$\text{Gini}(p)=1-\sum_{i=1} P(\frac{i}{p})2$$

Avec p un patron dans l'ensemble d'apprentissage,

$P(\frac{i}{p})$ la probabilité d'obtenir au hasard un patron de classe i de l'ensemble d'apprentissage.

Les auteurs ont défini GiniL et GiniR correspondant au coté gauche et droit de la surface de décision. Cet hyperplan est calculé avec la mesure d'impureté correspondant à :

$$\text{Impuretée(p)}=\frac{P_L*GiniL(P_L)+P_R*GiniR(P_R)}{n}$$

Les calculs de la minorité maximale se font avec le calcul de la minorité :

$$\text{Minorité(p)}=\sum_{i=1}^{Max(p)}P_i$$

Avec $P_i$ est la proportion d'éléments de l'ensemble d'apprentissage a la position p qui vont en position $P_i$.

De même que pour la fonction de Gini, les auteurs ont défini MinoriteR et MinoriteL d'où la mesure :

MaxMinorité= Max(MinoritéL MinoritéR)

OC1 utilise la complexité du coût de l'élagage comme critère d'élagage par défaut. Cette méthode exige un positionnement séparé de l'élagage. L'idée de cette approche est de créer un ensemble d'arbres de taille décroissante par rapport à l'arbre initial.

Tous ces arbres sont employés aux fins de classifier le positionnement élagué. Le coût d'élagage permet alors de choisir le plus petit arbre dont l'exactitude est dans les limites des erreurs de la meilleure exactitude obtenue

## 3.5 Intégration des algorithmes d'apprentissage

### 3.5.1 Introduction

Comme déjà présenté, il existe plusieurs techniques pouvant être utilisées pour l'estimation de la qualité du logiciel. Dans cette section, nous allons introduire l'outil que nous avons conçu tout au début de notre recherche afin de regrouper l'ensemble des algorithmes d'apprentissage utilisés pour nos expérimentations dans une même interface. Il s'agit du système Mltools (Machine Learning Tools).

Mltools a été conçu dans le but d'offrir un outil souple pour manipuler une variété d'algorithmes d'apprentissage dans l'environnement Windows.

### 3.5.2 Description du système Mltools

Mltools doit recevoir en entrée le fichier initial qui représente l'ensemble des données. La figure 3.5 présente les composantes principales de ce système.

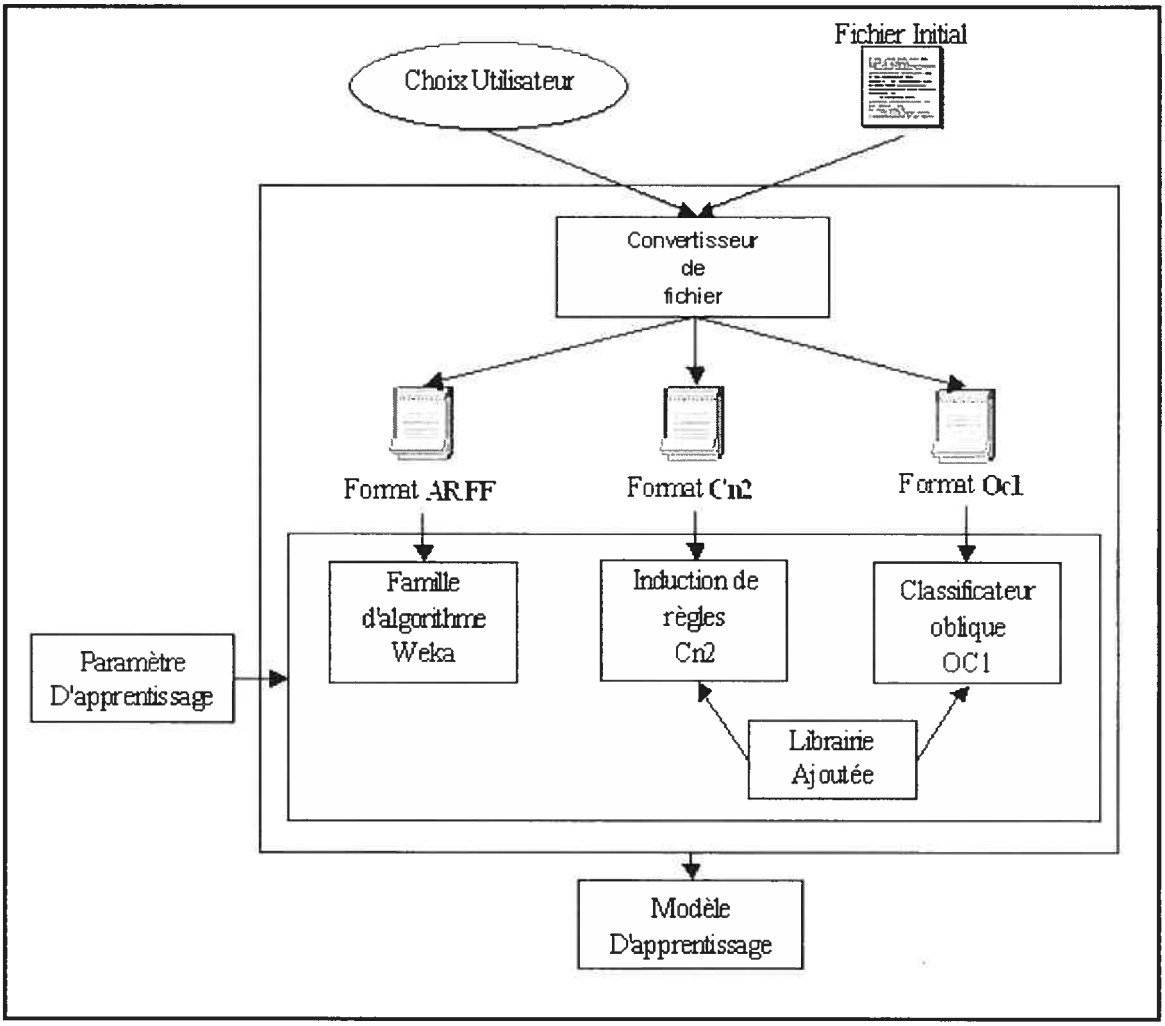


**FIG. 3.5 : Le système Mltools**

Comme nous pouvons le constater, le système Mltools se constitue essentiellement des éléments suivants:

- Module de conversion de fichier.
- Famille d'algorithmes WEKA.
- Induction de règle CN2.
- OC1.
- Bibliothèque de fonctions Unix.

La famille d'algorithmes weka, l'induction de règle CN2 et le classificateur oblique OC1 ont été présentés au début de ce chapitre. Dans ce qui suit, nous

allons présenter le module de conversion de fichiers et la bibliothèque de fonctions Unix

### 3.5.2.1 Module de conversion de fichiers

Une des étapes essentielles dans le processus d'apprentissage automatique, consiste en la préparation des données et c'est le module de conversion de fichiers qui se charge de cette tâche. Ce module reçoit en entrée le choix de l'utilisateur et selon ce choix on définit la transformation adéquate a faire :

-lire choix

Cas 1 : Convertir au format Arff

Cas 2 : Convertir au format CN2

Cas 3 : Convertir au format OC1

Selon l'algorithme d'apprentissage sélectionné par l'utilisateur, ce module a donc pour rôle principale de convertir le fichier initial aux trois formats suivants :

**- Format de données pour weka :**

Tous les algorithmes appartenant au système weka requièrent le même format :le format Arff. Pour représenter les données dans le format Arff, le module de conversion de fichier doit effectuer trois opérations :

- La première opération à effectuer consiste à placer le nom de la relation en début de fichier. Il suffit de faire procéder le nom de « @relation ».
- La deuxième opération consiste à lister les différents attributs de la relation de manière à ce que chaque attribut soit précédé du mot clé

« @attribute » et qu'il soit suivi de son nom et des différentes valeurs qu'il peut prendre : @attribute <nom><type>.

- La troisième et dernière opération consiste à introduire toutes les données en commencant par écrire« @data ».. Les données sont séparées par des virgules, et des points d'interrogation représentent les valeurs manquantes. Dans notre cas nous n'avons pas de valeurs manquantes.

Voici un exemple de fichier au format Arff généré par le module de conversion de fichier :

```
Sunny      85   85   FALSE   no
Sunny      80   90   TRUE    no
Overcast   83   86   FALSE   yes
Rainy      68   80   FALSE   yes
Rainy      65   70   TRUE    no
```

**FIG.3.6: Fichier source à convertir**

```
@relation weather

@attribute outlook {sunny,
overcast, rainy}
@attribute temperature real
@attribute humidity real
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}

@data
sunny,85,85,FALSE,no
sunny,80,90,TRUE,no
overcast,83,86,FALSE,yes
rainy,68,80,FALSE,yes
rainy,65,70,TRUE,no
```

**FIG. 3.7: Exemple de fichier Arff**

**- <u>Format de données pour CN2</u>**

Pour l'algorithme d'apprentissage CN2, le module de conversion de fichiers doit fournir deux ensembles en entrées :

1) L'ensemble des déclarations d'attributs.
2) L'ensemble des exemples.

Notons que ces deux ensembles peuvent être mis dans le même fichier, ou bien dans deux fichiers distincts. Dans notre cas, le module de conversion de fichier génère un seul fichier (contenant la déclaration des attributs et les exemples).
Pour représenter les données dans un format que CN2 peut lire, ce module doit :

- Placer l'entête,' **ATTRIBUTE AND EXAMPLE FILE** ', au début du fichier, suivi de la déclaration des attributs.
- La déclaration des attributs consiste à lister les différents attributs :On doit commencer par le nom de l'attribut ensuite les différentes valeurs qu'il peut prendre :

  <nom-attribut > :<différentes valeurs de l'attribut>;

  ou sinon <nom-attribut > :<type-attribut>.
- La troisième et dernière opération consiste à introduire toutes les données en commencant par écrire « @ », pour séparer entre la déclaration des attributs et celle des exemples.

Les données sont séparées par des espaces (au minimum un espace ).
Voici un petit exemple du format requis pour l'algorithme d'apprentissage CN2 :

```
**ATTRIBUTE AND EXAMPLE FILE**

cyclomatic_complexity: (INT)
statements: (INT)
executable: (INT)
declarative: (INT)
total_source_line: (INT)
Class: f d ;

@
197  363  285  78  1009    f;
0    201  0    201 694     f;
159  331  195  136 1300    f;
```

**Fig. 3.8 : Exemple de fichier requis pour CN2**

**- Format de données pour OC1**

Pour le format de données de OC1, c'est un format très simple. Il suffit de mettre toutes les données dans le même fichier et de séparer les attributs par un espace. Aucune déclaration d'attributs n'est requise.

### 3.5.2.2 Bibliothèque Unix

Pour créer notre système MLTools (Machine Learning Tools), nous avons utilisé le compilateur du visual C++ 6 de Microsoft [16],[52].

Comme nous l'avons déjà cités en 3.2 , le système weka est écrit en java ce qui veut dire qu'il est portable sur toute les plates formes. Son intégration dans MLT n'a pas posé de problème. Par contre, en ce qui concerne les deux algorithmes d'apprentissages : CN2 et OC1, ces derniers étant écrit en C ANSI, pour Unix, certaines difficultés sont apparues au niveau du compilateur de visual C++ .
Un des principaux problème réside dans le modèle de processus. Unix possède le processus fork, ce qui n'est pas le cas de Windows. D'autres fonctions aussi comme :Sys_call() sont méconnues par le compilateur de Visual C++.

Les applications que nous avons migrée faisant beaucoup appels à ce genre de fonctions, nous avons essayé de chercher les fonctions VC++, qui pouvait faire le même travail que ces fonctions manquantes. Dans certains cas, c'était impossible.

Pour pallier à ce problème de portabilité des applications de Unix vers Windows, nous avons du rajouter une bibliothèque Unix : Cette solution

consiste à regrouper l'ensemble des fonctions Unix manquantes dans un même environnement et le partager aux applications OC1 et CN2.
Ce genre de fonctions est disponible sur Internet. Elles ont été créé dans l'objectif de remédier au problème de migration.

Voici une capture d'image représentant notre Système MLTools (Voir FIG. 3.8)

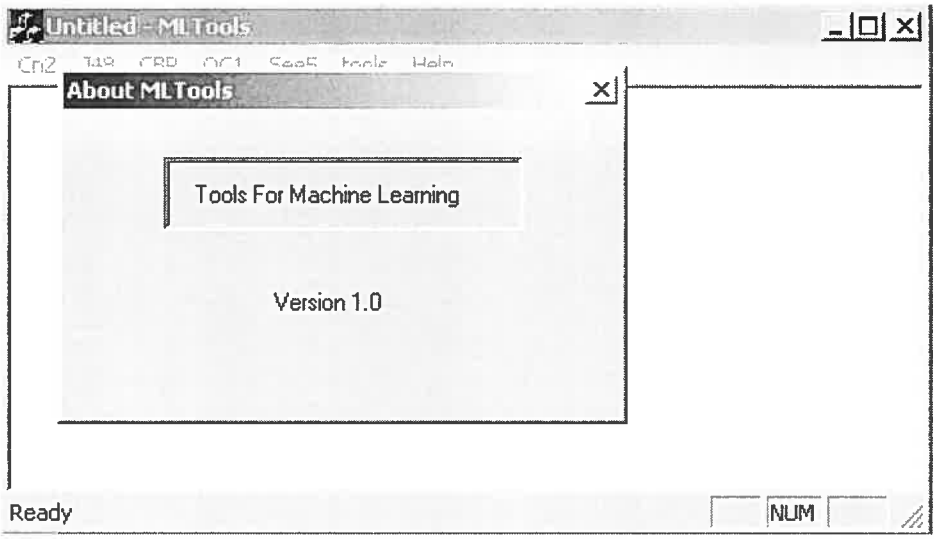


**Fig 3.9: Capture d'image de l'interface Mltools.**

# Chapitre 4

# Problèmes de prédiction / estimation étudiés

## 4.1 Généralités

Afin de souligner l'importance de l'utilisation des algorithmes d'apprentissage pour la production de modèles de prédiction, nous allons travailler sur la vérification d'hypothèses impliquant trois attributs de qualité (qui ne sont pas mesurables à priori mais directement exploitables). Pour effectuer cela, nous utiliserons des mesures internes de produit logiciel prélevées sur des composants logiciels orientées objets (complexité, couplage, cohésion, héritage, etc.).

Pour chaque expérience, nous allons suivre les étapes suivantes :

- Proposition des hypothèses reliant des propriétés de conception de composants orientés-objet avec une qualité de logiciel à prédire.
- Liste des mesures appropriées utilisées pour la prédiction.

Nous allons essayer de démontrer l'existence d'un lien causal entre mesures logiciels et attributs de qualité en mettant en évidence l'apport des mesures pour l'évaluation des attributs de qualité des logiciels comme le stipule le standard ISO/IEC numéro 14598 : « les mesures d'attributs internes sont utiles surtout si elles sont reliées à des caractéristiques externes de qualité ».

## 4.2 Problème 1 : Estimation des coûts de la maintenance corrective

### 4.2.1 Description du problème

L'ensemble des activités de maintenance est lié aux changements, aux modifications ou à d'autres facteurs qui peuvent altérer les logiciels nécessaires pour garder un système opérationnel après son approbation et sa mise en production.

Afin de réduire les coûts de maintenance, on doit trouver le moyen d'estimer et d'améliorer la maintenabilité du produit logiciel. Un produit est maintenable s'il est malléable aux modifications, lesquels ne doivent pas engendrer des conséquences imprévues. Par là même, il offre la possibilité de retracer l'origine d'un problème.

Une fois le logiciel livré, de nouveaux besoins peuvent se présenter dans le futur, par exemple une demande de correction ou l'ajout d'une nouvelle fonctionnalité. Il est évident qu'il est difficile d'évaluer et de caractériser la maintenabilité d'un produit à priori mais avec les différentes expériences du passé, cette tâche devient moins ardue. Des développeurs expérimentés peuvent se référer à leur expérience pour donner une estimation des efforts à fournir pour la maintenance. Cette méthode étant informelle et non généralisable, nous allons dans ce sens essayer de vérifier la première hypothèse (hypothèse H1) qui concerne l'importance de certaines mesures internes de logiciel pour la prédiction du coût de la maintenance corrective.
Il s'agira de trouver les modèles permettant de mieux estimer la maintenabilité du produit logiciel dans le but d'évaluer la charge à attribuer à la maintenance corrective qui est la partie consacrée à la correction des erreurs.

En d'autres termes, nous devons être capable de prédire si un logiciel sera «facile» ou «difficile» à maintenir dans le futur, ceci, en se basant sur ces mesures internes.

### 4.2.2 Données logicielles

Pour cette expérience, nous disposons d'un jeu de données Ada. Il s'agit d'un ensemble de données décrivant la maintenance d'une librairie de composants réutilisables de la NASA [3]. Le développement des composants a commencé en 1993. Ce sont des composants Ada83 totalisant 515000 lignes de code source.

Ces composants comptent 164 entrées structurées en 18 attributs comme la complexité cyclomatique, le nombre total de lignes de source, etc. (Voir TAB. 5.2). Chaque entrée est étiquetée par l'attribut de classe (maintenance) qui est à valeur binaire : F (ou 0) pour dire que la maintenance est facile et D (ou 1) pour prédire une maintenance difficile.

Un suivi de la maintenance des composants ADA a permis de collecter les erreurs et les failles rencontrées dans cette librairie. On désignera par erreur tout formulaire de demande de changement et par faille un composant caractérisé par un changement physique.

Sachant que la variable à prédire est définie comme l'effort déployé pour isoler et corriger un composant défaillant, l'effort d'isolation et de correction est mesuré selon les 4 échelles suivants :

- Une heure
- Une heure à un jour
- Un jour jusqu'à trois jours
- Plus que trois jours

Une fois l'erreur repérée, les mainteneurs identifient les causes et localisent les modifications à apporter. Par la suite, les changements de design, de code et de documentation seront effectués. Seront également fournis les noms des composants modifiés.
Dans notre étude, nous avons divisé le coût de la maintenance corrective en deux catégories :

- Maintenance facile : Si l'effort d'isolation et correction est inférieur à une journée.
- Maintenance difficile : Si l'effort d'isolation et correction dépasse une journée.

### 4.2.3 Hypothèse H1

L'hypothèse H1 concerne l'influence de certaines mesures internes sur le coût de la maintenance corrective. Ces mesures vont être présentées dans la section suivante.

#### 4.2.3.1 Variables (métriques) mises en jeu

Nous présentons ici l'ensemble des mesures internes utilisées pour la validation de l'hypothèse H1. Notons que l'extraction de ces mesures a été réalisée avec l'outil ASAP à partir de composants défaillants [24], [33], [40].

| | Attribut | Rôle |
|---|---|---|
| 1 | Ada language statements | Nombre d'instructions Ada. |
| 2 | Declarative | Nombre de déclarations dans une méthode |
| 3 | Statements | Nombre d'instructions dans une méthode. |
| 4 | Executable | Nombre d'instructions exécutables dans une méthode |

| | | |
|---|---|---|
| 5 | Total source lines | Nombre total des lignes incluant les lignes de code, les lignes de commentaires et les lignes vides. Cette mesure consiste tout simplement à compter le nombre de lignes du code (LOC). Signalons qu'il faut faire la distinction entre les lignes commentées (CLOC) et les lignes non commentées (NLOC). La longueur totale d'un programme LOC = NLOC + CLOC. Le rapport CLOC/LOC indique le niveau de 'commentaires' du programme. Ces mesures ne donnent qu'une faible indication de la complexité du programme. |
| 6 | Cyclomatic complexity | Cette mesure a été développée en 1976 par Mc Cabe. Elle fournit une mesure quantitative de la complexité logique d'un logiciel. et permet de :<br>• Mesurer le nombre de chemins conditionnels d'un programme.<br>• Attribuer une valeur de complexité.<br>C'est en fait le nombre de chemins indépendants dans un graphe de flux de contrôle (autrement dit, le nombre de cas de tests à effectuer pour couvrir au moins une fois tous les énoncés). Mc Cabe affirme que les parties d'un logiciel dont la complexité cyclomatique est importante manifestent un risque d'erreur élevé. |
| 7 | Lines of code | C'est le nombre de lignes de code dans la méthode, incluant les lignes de code, de commentaires et vides. |
| | Maximum statement | Représente la profondeur maximale des instructions |

| | | |
|---|---|---|
| 8 | nesting depth | emboîtés pour la totalité des méthodes. La profondeur d'une instruction simple est de 0. Pour les instructions emboîtées, la valeur augmente de 1 pour chaque nouvelle imbrication. |
| 9 | Lines of comments | C'est le nombre de lignes de commentaires |
| 10 | Total statement nesting depth | C'est la somme de la totalité des profondeurs |
| 11 | Inline comments | Ce sont les commentaires qui apparaissent sur la même ligne que ce qu'ils décrivent. Il sont très utiles pour expliquer brièvement avec quelques informations les particularités du code. |
| 12 | Average statement nesting depth | C'est la moyenne des profondeurs des instructions emboîtées |
| 13 | Blank lines | Nombre de lignes vides |
| 14 | Number of distinct operators (n1) | Nombre d'opérateurs distincts |
| 15 | Number of distinct operands (n2) | Nombre d'opérandes distincts |
| 16 | Number of operators (N1) | Nombre total d'opérateurs utilisés |
| 17 | Number of operands (N2) | Nombre total d'opérandes utilisés |

**TAB. 4.1 : Mesures de maintenance corrective**

Les quatre mesures primitives : n1, n2, N1, N2 ont été définies par Halstead [34]. Ces mesures supposeraient qu'un programme est constitué uniquement d'opérateurs et d'opérandes.

## 4.3 Problème 2 : Prédiction de la réutilisabilité

### 4.3.1 Description du problème

La réutilisation des composants orientés-objets permet de diminuer le coût et la durée du développement des logiciels tout en offrant une meilleure qualité. Elle ne se limite pas seulement à éviter de réécrire du code mais à faciliter considérablement la maintenabilité, la fiabilité, l'évolutivité, l'utilisabilité et l'adaptabilité car elle se base en général sur des composants testés et ayant déjà fait leurs preuves [23].

La réutilisabilité nous permet d'éviter de rechercher des solutions à des problèmes déjà rencontrés et résolus en exploitant certaines parties de codes de logiciels déjà existants et qui ont des points communs avec le logiciel à bâtir.

Notre souci est d'automatiser la détection des composants potentiellement réutilisables. Nous partons avec l'idée que certains attributs internes tel que l'héritage, le couplage et la complexité peuvent constituer de bons indicateurs sur la possibilité de réutilisation d'un composant. Établir une relation directe entre ces attributs (qui sont mesurables) et le potentiel d'un composant d'être réutilisable (qui n'est pas directement mesurable) peut être une voie intéressante pour automatiser la détection.

La réutilisabilité est un facteur complexe. Certains composants sont plus réutilisables dans un domaine que dans d'autres. Notre objectif n'est pas de chercher un ensemble de méthodes permettant de mesurer la réutilisabilité d'une façon universelle mais plutôt d'étudier certains aspects et caractéristiques spécifiques concernant les langages de programmation orientés-objets (comme le C++ ) qui affectent la réutilisabilité.

Nous proposons pour la réutilisabilité quatre (4) hypothèses (H2a, H2b, H2c et H2d) qui concernent les rapports entre la réutilisabilité et, respectivement, l'héritage, le couplage (au niveau code et conception) et la complexité.

Des aspects différents peuvent être pris en considération pour mesurer empiriquement la réutilisabilité d'un composant dépendamment du point de vue adopté. Par exemple, nous pouvons considérer que la réutilisabilité représente la quantité d'efforts nécessaires pour réutiliser un composant d'une certaine version d'un système dans une autre version du même système. Ou alors on peut considérer que la réutilisabilité représente la quantité d'efforts nécessaires pour réutiliser un composant d'un système dans un autre ayant le même domaine d'application [44].

Pour nos expériences, nous avons adopté ce dernier aspect afin de définir la mesure de réutilisabilité.

Nous essayerons de dégager les propriétés du code qui affectent la réutilisabilité et établirons des modèles prédictifs qui mettent en relief la relation entre cette qualité et les mesures internes. Sachant qu'un composant est réutilisable si l'effort déployé pour sa réutilisation est inférieur à l'effort d'un nouveau développement, il s'agira de trouver l'algorithme d'apprentissage le plus efficace pour la détection de tels composants orientés-objets réutilisables.

### 4.3.2 Données logicielles

L'échantillon de données sur lequel nous avons travaillé comprend 84 classes. L'extraction de ces mesures a été réalisée avec deux outils : il s'agit de GEN++ développé par Prem Devanbu [6], [7] et son équipe et QMOOD (Quality Model for Object-Oriented Designs) [5]. Ces outils ont la capacité d'extraire des informations relatives à l'organisation des programmes comme la hiérarchie des classes et la complexité des classes.

Notons que la variable dépendante (Reusability) représente ici la quantité d'effort nécessaire pour réutiliser un composant d'un système dans un autre ayant le même domaine d'application. Elle est d'autant plus grande que la réutilisabilité est moins importante. Pour les différentes classes étudiées, les valeurs attribuées à la réutilisabilité sont [8] :

- Totalement réutilisable : il s'agit d'un composant générique à un certain domaine; nous attribuons la valeur 1.
- Réutilisable avec un minimum d'effort : moins de 25% du code nécessite des modifications pour être utilisé dans le nouveau système; dans ce cas nous attribuons la valeur 2.
- Réutilisable avec un effort important : plus de 25% du code nécessite des modifications pour être utilisé dans le nouveau système; nous attribuons la valeur 3.
- Non réutilisable: signifie que le composant est très spécifique au système; nous attribuons la valeur 4.

### 4.3.3 Hypothèse H2a

L'héritage est une manière naturelle de modéliser le monde réel (le domaine d'application). Ce processus de modélisation est implanté sur le paradigme objet à travers une structure hiérarchique de classes basée sur une relation transitive et antisymétrique *entre classes* qui s'appelle héritage.

Deux classes différentes peuvent partager partiellement la description de leurs états et de leurs comportements. Pour éviter les redondances de description et faire ressortir les différences entre ces classes dans une structure hiérarchique d'héritage, la notion de super-classe est introduite (classe-mère) pour rassembler les descriptions communes et de sous-classe (classe-fille ou simplement classe) pour décrire les spécificités.

Dans un système orienté-objets nous avons deux (2) types de classes :

• Des classes générales: Ce genre de classe contient des objets et des méthodes générales pour fournir des services à d'autres classes.
• Des classes spécifiques : Ces classes traitent des opérations spécifiques et elles sont utilisables par un système spécifique.

Les classes générales sont disposées à un niveau hiérarchique élevé tandis que les classes spécifiques sont placées à un niveau plus bas. Les classes placées au sommet de la hiérarchie sont aptes à être réutilisées tandis que celles qui sont au niveau inférieur, dépendent fortement des classes au niveau supérieur, ce qui engendre une compléxité à la compréhension et une grande difficulté à les isoler. Ce genre de classe ne sont guère candidates à la réutilisation [54].

Nous allons donc vérifier la validité de l'hypothèse suivante :

**Hypothèse H2a**

*La position d'un composant dans une hiérarchie de classes du système affecte d'une certaine manière sa réutilisabilité.*

#### 4.3.3.1 Variables (métriques) mises en jeu

- Depth Of Inheritance Tree (DIT)
- Height Of Inheritance Tree (HIT)
- Number Of Ancestors (NOA)
- Number Of Children (NOC)
- Number of descendants (NOD)
- Number Of Inherited Methods (OIM)
- Number Of Parents (NOP)

Ces métriques ont été définies au chapitre 2.

### 4.3.4 Hypothèse H2

Le degré d'interdépendance entre les composants d'un système orienté-objets peut être mesuré grâce au couplage. Deux classes sont dites couplées si une méthode de l'une utilise une méthode ou un attribut de l'autre. Pour aboutir à une bonne qualité de conception et d'implémentation, le couplage entre classes devrait être minimal [45].
Plus une classe est couplée à d'autres classes, plus il serait difficile de la réutiliser.

Nous essayerons de valider l'hypothèse suivante qui exprime les idées que nous venons de citer :

**Hypothèse H2**

*Les dépendances entre un composant et son environnement affecte d'une certaine manière sa réutilisabilité.*

La réutilisation des composants à fort couplage nécessite beaucoup d'effort de compréhension, de test et de vérification avant l'étape d'intégration dans un nouveau système. Aussi, les risques d'erreurs sont très grands [40]. Il n'est donc pas intéressant de réutiliser de tels composants car tenter de les comprendre nécessite la compréhension d'autres composants dans le système.

Deux types de mesures de couplage seront utilisés pour vérifier cette l'hypothèse:

- Mesures de couplage au niveau conception.
- Mesures de couplage au niveau code.

Nous aurons donc à valider deux hypothèses : H2b et H2c

#### 4.3.4.1 Hypothèse H2b

**Mesures de couplage au niveau conception**

Briand et Al [11] ont proposé les mesures suivantes afin de mesurer le niveau de couplage d'une classe durant la conception de systèmes orienté-objets. Cet ensemble de mesures représente différents aspects du langage C++ (spécialisation, agrégation, relations d'amitié entre classes, etc.)

##### 4.3.4.1.1 Variables (métriques) mises en jeu

La liste des mesures suivantes, a été proposée par [10]. Ces mesures prennent en compte différents mécanismes du langage C++ comme la spécialisation, agrégation, etc.

- Inverse Friend Class-Attribute Import Coupling (IFCAIC)
- Ancestors Class-Attribute Import Coupling (ACAIC)

- Others Class-Attribute import coupling (OCAIC)
- Friends Class Export Coupling (FCAEC)
- Descendants Class-Attribute export coupling (DCAEC)
- Others Class-Attribute Export Coupling (OCAEC)
- Inverse Friend Class-Method Import Coupling (IFCMIC)
- Ancestors Class-Method import coupling (ACMIC)
- Others Class-Method import coupling (OCMIC)
- Friends Class-Method Export Coupling (FCMEC)
- Descendants Class-Method export coupling (DCMEC)
- Others Class-Method Export Coupling (OCMEC)
- Inverse Friend Method-Method Import Coupling (IFMMIC)
- Ancestors Method-Method Import Coupling (AMMIC)
- Inverse Friend Method-Method Export Coupling (IFMMEC)
- Others Method-Method Import Coupling (OMMIC)
- Others Method-Method Export Coupling (OMMEC)
- Ancestors Method-Method Export Coupling (AMMEC)

#### 4.3.4.2 Hypothèse H2c

Nous allons valider l'hypothèse H2 mais cette fois en prenant en considération les mesures de couplage au niveau code (H2c). La définition des mesures de couplage utilisées dans cette expérience a été proposée par Lounis et Al [38] avec pour objectif de mesurer la présence du couplage au niveau code. Voici quelques une de ces mesures :

##### 4.3.4.2.1 Variables (métriques) mises en jeu

- No Parameter Interconnection (NPI)
- Scalar-Data Interconnection (SDI)

- Return-Date Interconnection (RDI)
- Stamp-Date Interconnection (StDI)
- Scalar-Control Interconnection (SCI)
- Return-Control Interconnection (RCI)
- Stamp-Control Interconnection (StCI)
- Scalar-Data/Control Interconnection (SDCI)
- Return-Data/Control Interconnection (RDCI)
- Stamp-Data/Control Interconnection (StDCI)
- Tramp Interconnection (TI)
- Scalar-Reference Data Interconnection (SRDI)
- Scalar-Reference Control Interconnection (SRCI)
- Scalar-Reference Data-Control Interconnection (SRDCI)
- Scalar-Reference Modification Interconnection (SRMI)
- Stamp-Reference Data Interconnection (StRDI)
- Stamp-Reference Control Interconnection (StRCI)
- Stamp-Reference Data-Control Interconnection (StRDCI)
- Stamp-Reference Modification Interconnection (StRMI)
- Global-Data Interconnection (GDI)
- Global-Control Interconnection (GCI)
- Global-Modification Interconnection (GMI)
- Type Interconnection (TyI)

### 4.3.5 Hypothèse H2d

La complexité influence la réutilisation sur 3 points. Premièrement, avant d'extraire un composant pour le modifier et ensuite le réutiliser, on doit bien le comprendre (le coût de réutilisation est fortement influencé par sa compréhension). Plus le composant est complexe, plus il est difficile de le comprendre [42].

Deuxièmement, une classe avec un grand nombre de méthodes a beaucoup de chance d'être spécifique, ce qui la rend moins réutilisable. Finalement, un composant ayant un code compliqué requiert un grand niveau de compréhension dans la partie test et déboguage de la classe. Dans ce cas, le coût d'utilisation d'un tel composant est élevé et rend sa modification et son intégration difficiles.

Les auteurs [5], [42] affirment que les mesures de complexité et de volume (taille) sont aptes à prédire l'effort de maintenance ainsi que le coût d'adaptation pour réutiliser les composants logiciels. Le coût de réutilisation d'une classe, incluant son extraction et sa réadaptation, est fortement influencé par sa compréhension et peut être évalué en mesurant son volume et sa complexité.

Pour la suite, on va énoncer une hypothèse relative à l'impact de la complexité sur la réutilisabilité.

**Hypothèse 2d**

*Le volume et la complexité d'un composant affectent d'une certaine manière sa réutilisabilité.*

#### 4.3.5.1 Variables (métriques) mises en jeu

Pour cette expérience, nous avons utilisé les mesures suivantes (leurs définitions figurent au niveau du chapitre 2):

- Weighted Methods Per Class (WMC)
- Response For a Class (RFC)
- Number Of local Methods (NOM)
- Class Interface Size (CIS)

- Number Of Parametres Per Method (NPM)
- Number of Polymorphic Methods (NOPM)
- Number Of Abstract Data Types (NAD)
- Number of Reference Attributes (NRA)
- Number of Public Attributes (NPA)
- Class Size in Bytes (CSB)

## 4.4 Problème 3 : Estimation de la propension à engendrer des erreurs

### 4.4.1 Description du problème

De plus en plus d'entreprises dépendent crucialement de leur système informatique. Ainsi, le fonctionnement correct du système est nécessaire pour assurer un fonctionnement correct et optimal de l'entreprise. C'est la raison pour laquelle de nombreux travaux de recherches se sont concentrés sur l'étude des composants logiciels pouvant contenir des erreurs dans le but d'identifier les actions les plus appropriées permettant d'optimiser le processus de développement de logiciel.

Certaines études menées par [19] et [61], ayant pour objectif l'amélioration du produit logiciel en termes de prédiction des défaillances, ont essayé d'identifier les erreurs très tôt durant la phase de conception du système. Leur souci était de réduire les coûts de la maintenance dans le futur.

Certaines caractéristiques de conception de classes orientées-objets, internes (comme la cohésion) et externes (comme le couplage et l'héritage), influent sur l'attribut de qualité que nous allons étudier au niveau de cette expérience. Il s'agit de la propension à engendrer des erreurs.

La propension à engendrer des erreurs est un attribut de qualité de logiciel qui peut être influencé par plusieurs facteurs comme l'environnement, l'expérience des développeurs, les processus, les méthodes, les outils utilisés et la formation [56].
L'objectif de notre recherche peut se résumer en deux (2) points :

- Identifier et valider les mesures qui sont de bons outils de prédiction de risques.
- Déterminer le modèle de prédiction adéquat pour la détection de la propension à engendrer des erreurs.

Nous devons donc trouver un lien causal reliant la propension à engendrer des erreurs avec les mesures de conception que l'on va proposer pour notre étude.

### 4.4.2 Données logicielles

Pour réaliser cette étude, nous avons exploité les données provenant du système LALO (Langage d'Agents Logiciels Objet) [68]. Ce système a été développé au CRIM (Centre de Recherche Informatique de Montréal) depuis 1993. Le système LALO comprend environ 90 composants C++, totalisant environ 57.000 lignes source.
Dans le cadre de notre expérience, nous avons utilisé uniquement les 83 classes développées par l'équipe LALO. Les données extraites de ces composants concernent les mesures liées à l'héritage, à la cohésion et au couplage ainsi que divers attributs de données présentant des défaillances.

Notre objectif est de construire un modèle prédictif permettant d'identifier le composant pouvant contenir des erreurs. Pour cela, on définit l'erreur comme étant une action humaine causant une 'faute' dans un produit logiciel ; ce qui

engendre des défaillances du produit. L'opération qui consiste à éliminer une faute est appelée correction [35].

D'après la terminologie de l'IEEE (norme 729) *« la faute est à l'origine de l'erreur qui se manifeste par des anomalies dans le logiciel qui peuvent causer des pannes ».*

L'attribut à prédire dans cette expérience se nomme 'erreur'. On la définit comme étant *'le nombre de corrections effectuées'.*
Pour les différentes classes retenues, les valeurs attribuées à la variable à prédire erreur sont [35]:

- On attribue la valeur 0 à la variable erreur pour souligner que le composant ne contient pas d'erreurs.
- On attribue la valeur 1 à la variable erreur pour souligner que le composant a de faible chance de contenir des erreurs.
- On attribue la valeur 2 à la variable erreur, pour souligner que le composant peut contenir des erreurs.

Notons que dans une étude faite par Lounis et Al [3], une série de mesures a été proposée et validée empiriquement à travers un modèle prédictif traitant de la propension à engendrer des erreurs en utilisant C4.5.

### 4.4.3 Hypothèse H3a

L'héritage est un élément essentiel dans un système orienté-objet. En effet, l'héritage définit les relations entre classes, là où une classe partage la structure et le comportement d'une ou de plusieurs classes (cas de l'héritage simple ou multiple).

Il paraît logique d'affirmer que plus une sous-classe est placée en bas de la hiérarchie, plus elle a des possibilités d'héritage. De ce fait, elle a tendance à capter davantage d'erreurs. D'autres part, plus une classe erronée a de descendants, plus grand sera le nombre de sous-classes affectées à cause de l'héritage. Par exemple, s'il y a plusieurs sous-classes qui dépendent des méthodes et des attributs définis dans la classe supérieure, n'importe quel changement apporté à ces méthodes ou attributs peut affecter les sous classes. Ainsi il devient très ardu de maintenir une classe [57].

Nous proposons l'hypothèse suivante :

**Hypothèse H3a**

*La position d'un composant dans une hiérarchie de classes affecte sa propension à engendrer des erreurs.*

#### 4.4.3.1 Variables (métriques) mises en jeu

Pour vérifier cette hypothèse, nous utiliserons la liste suivante, qui représente onze (11) mesures d'héritage, leurs définitions se trouvent au chapitre 2 :

- Depth of Inheritance Tree (DIT)
- Average inheritance depth (AID)
- Class to leaf depth (CLD)
- Number Of Children (NOC)
- Number Of Parents (NOP)
- Number Of Descendants (NOD)
- Number Of Ancestors (NOA)
- Number of Methods Overriden (NMO)
- Number of Methods Inherited (NMI)
- Number of methods Added, (new methods) (NMA)

- Specialization IndeX (SIX)

### 4.4.4 Hypothèse H3b

Briand et Chidamber [12], [17] soulignent qu'une bonne qualité de conception de logiciel obéit, entre autres, au principe de forte cohésion. La cohésion capture l'ampleur avec laquelle chaque groupe de déclarations de données et sous-programmes qui sont conceptuellement connexes appartiennent au même module.

Un degré élevé de cohésion est souhaitable dans un système et les déclarations de données et sous-programmes qui ne sont pas connexes doivent être encapsulés dans différents modules.
Stevens, Myers et Constantine sont les premiers à avoir introduit la cohésion dans le contexte des techniques de développement structuré. Ils définissent la cohésion comme étant le degré d'appartenance des éléments d'un module. Plus la cohésion d'un module est forte, plus il sera simple à développer, à maintenir et à réutiliser, et moindre sera la propension à engendrer des erreurs [12], [9].

Dans ce cadre, nous proposons l'hypothèse suivante :

**Hypothèse 3b**

*Le degré de cohésion d'un composant affecte sa propension à engendrer des erreurs.*

#### 4.4.4.1 Variables (métriques) mises en jeu

Les mesures de cohésion suivantes, dont la signification a été présentée au chapitre 2, ont été sélectionnées :

- Lack of cohesion in methods (LCOM1)

- Lack of cohesion in methods (LCOM2)
- Lack of cohesion in methods (LCOM3)
- Lack of cohesion in methods (LCOM4)
- Lack of cohesion in methods (LCOM5)
- Lack of cohesion in methods (Coh)
- Connectivity (Co)
- Loose Class Cohesion (LCC)
- Tight Class Cohesion (TCC)
- Information flow based cohesion (ICH)

### 4.4.5 Hypothèse H3c

Une bonne qualité de conception de logiciels obéit, entre autres, au principe de faible couplage [45], [9], [27], [60].

Plus le couplage est élevé entre les modules, plus il sera difficile de les comprendre, les changer et , par la même, de les corriger. Ainsi, le système logiciel résultant serait très complexe, ce qui devrait augmenter la propension à engendrer des erreurs.

Si un grand nombre de méthodes d'une classe peuvent être invoquées en réponse à un message reçu par un autre objet de cette classe, la compréhension de cette classe devient très compliquée. Çela implique qu'il y'a des chances que cette classe soit erronée.

Dans ce cadre, on propose l'hypothèse suivante :

**Hypothèse H3c**

*Le degré de dépendance entre composants et leur environnement de définition affecte la propension à engendrer des erreurs.*

#### 4.4.5.1 Variables (métriques) mises en jeu

Les mesures de couplage suivantes, dont la signification a été présentée au chapitre 2, ont été sélectionnées :

- Coupling Between Object classes (CBO)
- Response For Class (RFC)
- Message Passing Coupling (MPC)
- Information flow based coupling (ICP)
- Information flow based inheritance coupling (IH-ICP)
- Information flow based non-inheritance coupling (NIH-ICP)
- Data Abstraction Coupling (DAC)
- Inverse Friends Class-Attribute Import Coupling (IFCAIC)
- Ancestors Class-Attribute Import Coupling (ACAIC)
- Others Class-Attribute Import Coupling (OCAIC)
- Friends Class-Attribute Export Coupling (FCAEC)
- Descendants Class-Attribute Import Coupling (DCAEC)
- Others Class-Attribute Export Coupling (OCAEC)
- Inverse Friends Class-Method Import Coupling (IFCMIC)
- Ancestors Class-Method Import Coupling (ACMIC)
- Others Class-Method Import Coupling (OCMIC)
- Friends Class-Method Export Coupling (FCMEC)
- Descendants Class-Method Export Coupling (DCMEC)
- Others Class-Method Export Coupling (OCMEC)
- Inverse Friends Class-Method Import Coupling (IFMMIC)
- Ancestors Method-Method Import Coupling (AMMIC)
- Others Method-Method Import Coupling (OMMIC)
- Friends Method-Method Export Coupling (FMMEC)
- Descendants Method-Method Export Coupling (DMMEC)

- Others Method-Method Export Coupling (OMMEC)

Dans le chapitre suivant, nous allons vérifier les hypothèses que nous venons d'énoncer.

# Chapitre 5

# Vérification des hypothèses : Expérimentations

## 5.1 Introduction

Nous allons vérifier les hypothèses présentées dans le chapitre 4. Il est important de savoir qu'il n y'a pas de recette magique qui mène vers le meilleur résultat. Il s'agit en l'occurence de procéder à plusieurs expérimentations avec les différents algorithmes d'apprentissage (présentés au chapitre 3) et de choisir à chaque fois la meilleure configuration qui mène au meilleur score. Les résultats de nos expériences seront évalués grâce aux outils que nous allons présenter dans ce qui suit.

## 5.2 Sélection d'attributs

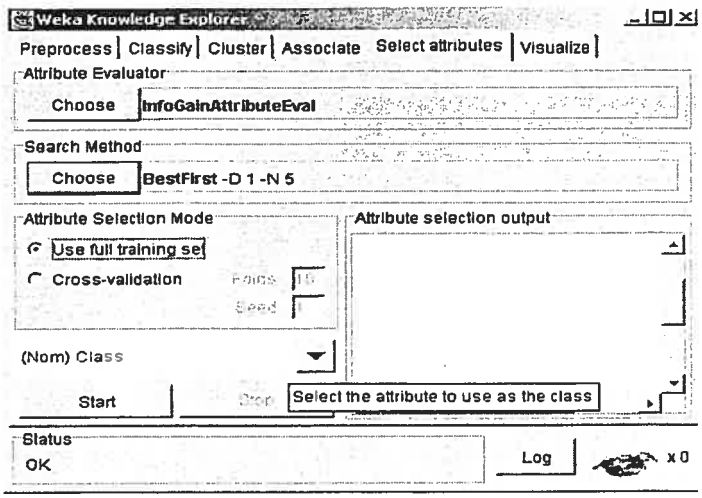


**FIG. 5.1: Sélection d'attributs dans WEKA**

La fenêtre se divise en plusieurs zones :

**Attribute Evaluator** : algorithme utilisé pour évaluer la pondération des attributs.
**Search méthode** : méthode de recherche utilisée.
**Liste déroulante** : sélection de l'attribut dont on veut prédire la valeur.

La sélection d'attributs nous permet d'avoir une idée sur les attributs les plus importants dans la prédiction des attributs de qualité. Grâce à cet outil, nous obtenons une liste ordonnée selon l'importance des poids accordés aux attributs. Plusieurs algorithmes d'évaluation et méthodes de recherche sont disponibles dans Weka. Nous avons remarqué que les meilleurs résultats étaient obtenus avec l'algorithme d'évaluation « Ranked » et la méthode de recherche « BestFirst ». Nous avons donc retenu ces derniers pour nos expériences.

## 5.3 Techniques de validation de modèles

Pour classifier un problème, il est naturel de mesurer les performances des classificateurs en terme de taux d'erreurs ou de succès. Le rôle du classificateur est de prédire la classe de chaque instance : Si elle est correcte, elle est admise comme succès. Dans le cas contraire, elle est comptabilisée en erreur.

Le taux d'erreurs est seulement une proportion d'erreurs faite à travers l'ensemble de toutes les instances. Cette mesure nous donne la performance du classificateur.

Pour établir les résultats de notre recherche nous allons exploiter les algorithmes de WEKA, OC1 et CN2 présentés dans le chapitre 3. Il ne s'agit pas de comparer entre ses algorithmes d'apprentissage, mais plutôt de faire le

maximum d'expérimentations afin de choisir celle qui donnera le meilleur résultat. Les résultats seront alors présentés dans un tableau de la forme suivante :

| | Train | Cross_validation | Split |
|---|---|---|---|
| OC1 | | | |
| CN2 | | | |
| J48 | | | |
| OneR | | | |
| Ibk | | | |
| Neural Network | | | |
| SVM | | | |
| Part | | | |
| M5' | | | |
| Naive Bayes | | | |

**TAB. 5.1: Résultat des expérimentations.**

La première colonne du tableau représente les algorithmes d'apprentissage utilisés dans nos tests (définis au chapitre 4). Au niveau de la deuxième colonne du tableau, nous allons enregistrer les performances du classificateur (algorithme d'apprentissage) en terme de taux de succès. Ce taux est évalué sur l'ensemble d'apprentissage et cette phase est nommée phase d'apprentissage (Train).

Dans cette phase, l'algorithme d'apprentissage, reçoit en entrée le training set (ensemble de toutes les données d'apprentissage) ainsi qu'un ensemble de paramètres d'apprentissages ($p_1$,$p_2$....$p_k$).

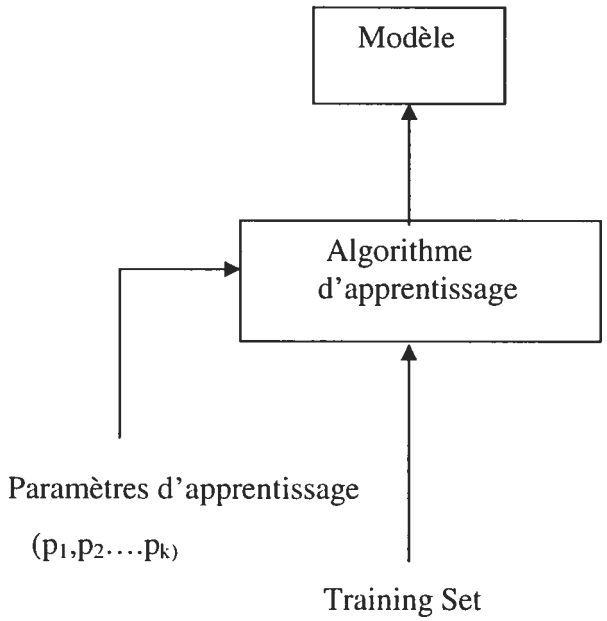


**FIG. 5.2: Entraînement d'un algorithme d'apprentissage**

Ce sont les données (training set) qui induisent l'apprentissage. Le résultat de cet apprentissage est appelé Modèle (Voir FIG. 5.2).

Il est important de savoir que l'on ne peut se fier aux données-résultats obtenus sur le training set car l'algorithme d'apprentissage a perdu son indépendance face à ces données. Il nous faut donc connaître le pouvoir de généralisation du classificateur, en fait sa capacité à obtenir de bons résultats par rapport à de nouvelles données que nous voudrons classer dans le futur. Ainsi, il nous faut évaluer les performances du modèle obtenu en se basant sur un ensemble indépendant de données : le test set.

Pour cela, nous allons utiliser les deux techniques d'évaluations suivantes (colonne 3 et 4 du tableau 5.1):

- Cross-Validation.
- Percentage Split.

### 5.3.1 Cross Validation

La cross-validation est une méthode qui permet d'estimer l'erreur sur l'ensemble de données qui n'ont pas été utilisées dans la phase d'apprentissage. Le résultat de l'erreur est souvent utilisé comme critère pour choisir le modèle le plus compétent pour un problème particulier.

Dans la K cross validation, les données sont divisées en K sous-ensembles (blocs de données) de même taille approximativement. Dans ce cas, la machine apprend K fois et à chaque fois elle enlève un des sous-ensembles de l'ensemble d'apprentissage. La moyenne des K erreurs estimée à chaque étape est prise comme estimation de l'erreur générale du modèle donné.

### 5.3.2 Percentage Split

Dans ce cas, le résultat de la classification va être évalué sur un ensemble de tests qui est une partie isolée de l'ensemble d'apprentissage originale. Le Split par défaut est de 66%. Ce qui signifie que 66% des données sont utilisés pour le training et 34% pour le test. À chaque résultat obtenu, nous allons préciser entre parenthèse le pourcentage du Split qui a été utilisé pour trouver le meilleur résultat.

Ces deux techniques sont efficace dans le cas ou l'on dispose pas de beaucoup de données. Alors on choisit le paramètre K et le pourcentage du Split de façon a séparer l'ensemble de données en ensemble d 'apprentissage et ensemble de test, en s'assurant que chaque classe est suffisamment représentée dans les deux ensembles et le taux de réussite du classificateur est mesurée par le taux de données bien classés.

## 5.4 Résultats expérimentation 1

Nous allons essayer de valider chacune des hypothèses présentées dans le chapitre 4. Nous commencerons par l'hypothèse H1:

### 5.4.1 Hypothèse H1

Dans cette partie, nous allons vérifier la validité de l'hypothèse H1 (voir § 4.2.2)

#### 5.4.1.1 Résultats et interprétations

**Résultat de la sélection d'attributs :**

Les paramètres utilisés dans cette sélection sont :

**Attribute Evaluator** : Ranked
**Search méthode** : BestFirst
**Liste déroulant** : Prédiction de la variable maintenance.

| Poids | Attribut |
|---|---|
| 54.268 | inline_comments |
| 52.439 | max_stat_nesting_d |
| 51.22 | blank_lines |
| 51.22 | Declarative |
| 50.61 | cyclomatique complexity |
| 49.39 | n1 |
| 48.78 | Executable |
| 48.171 | lines_of_Code |

| | |
|---|---|
| 48.171 | lines_of_Code |
| 48.171 | total_stat_nesting_dep |
| 48.171 | field19 |
| 47.561 | n2 |
| 47.561 | field20 |
| 46.951 | statements |
| 46.951 | ada_language_statements |
| 45.122 | lines_of_comments |
| 40.854 | average_stat_nesting_d |
| 40.854 | total_source_line |

**TAB. 5.2: Résultat de la sélection d'attributs pour l'hypothèse H1**

Le tableau 5.2, nous donne le classement des attributs selon l'ordre de pertinence décroissant.

On consigne dans les tableaux qui vont suivre, les meilleurs résultats obtenus.

**Taux de succès**

Pour chaque résultats concernant les colonnes 3 et 4 nous allons mettre, entre parenthèses, la valeur de K (colonne 3) et du pourcentage du Split (colonne 4)

.

| | Train | Cross_validation(**K**) | Split(**%**) |
|---|---|---|---|
| OC1 | 68.75% | 68.75%(10) | 49.5%(77) |
| CN2 | 86.72% | 73.33%(164) | 83.5%(78) |
| J48 | 54.26% | 54.26%(3) | 67%(79) |
| OneR | 70.73% | 58.53%(110) | 80.25%(72) |
| Ibk | 100% | 56.70%(15) | 56%(74) |

| | | | |
|---|---|---|---|
| Neural Network | 60.63% | 56.70%(7) | 60.22%(74) |
| SVM | 57.31% | 56.70%(14) | 50%(77) |
| Part | 60.36% | 52.43%(6) | 67.56%(78) |
| M5' | 56% | 58.09%(105) | 40.5%(74) |
| Naive Bayes | 53.65% | 54.87%(3) | 51.78%(66) |

**TAB. 5.3: Résultats Hypothèse 1 (H1)**

Les images suivantes présentent les résultats du tableau 5.3 sous forme d'histogrammes :

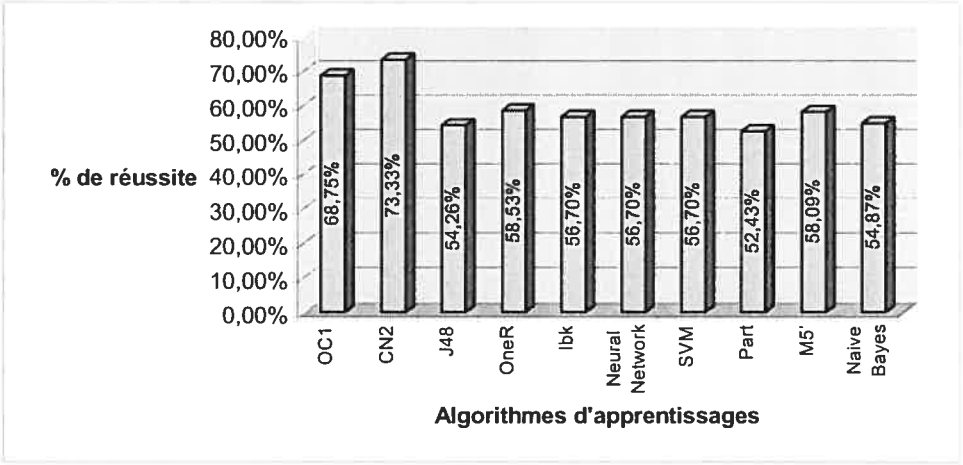


**FIG. 5.3: Histogramme hypothèse H1(cross-validation)**

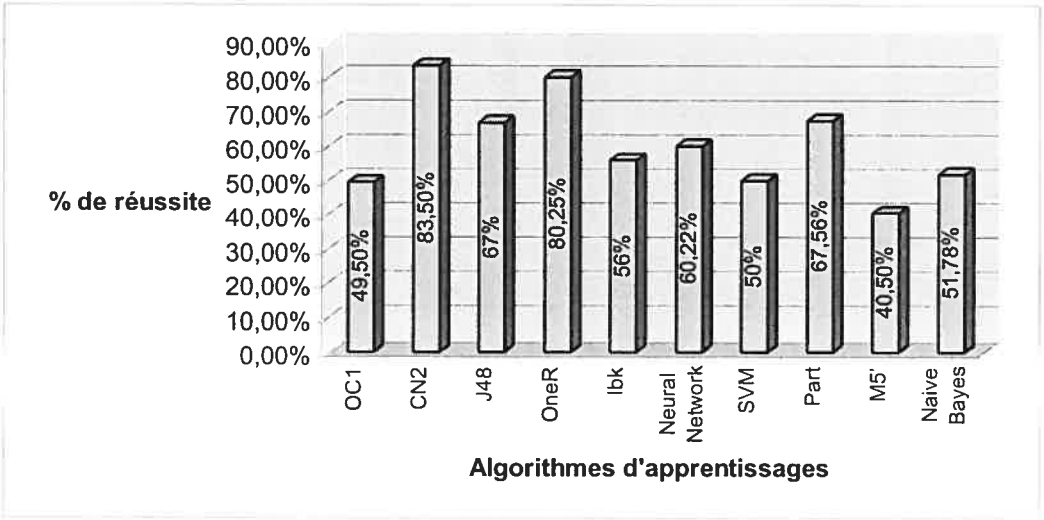


**FIG. 5.4 : Histogramme hypothèse H1(Split)**

Les résultats obtenus confirment la possibilité de l'existence d'un lien causal entre les mesures sélectionnées et la variable à prédire avec un degré de confiance d'environ 74% pour la cross validation et avec un degré de confiance de 83.5% concernant la technique du Split.

Voici un extrait de quelques règles produites par CN2 pour prédire si la maintenance sera facile ou difficile :

| | |
|---|---|
| Règle 1 :<br>Declarative <34.50<br>Line_of_code<451<br>→ maintenance 'f' | Règle 2 :<br>Si Total_source_line<1306.50<br>et **Inline_comments** >50.50<br>→ maintenance 'f' |
| Règle 3 :<br>Si Total_source_line<374.50<br>et n1<18.50<br>→ maintenance 'd' | Règle 4 :<br>Si Cyclomatic_complexity>19.50<br>et n2 <132<br>→ maintenance 'd' |

**TAB. 5.4: Règles générées par CN2(hypothèse H1)**

Si nous considérons par exemple la règle 3 du modèle, nous pouvons la traduire par : « si le 'total_source_line' est inférieur à 1306.50 et que le nombre de commentaires('Inline_comments') est supérieur à 50.50, dans ce cas, on prédit une maintenance facile ».

Par contre la règle 5 spécifie que : « Si la complexité cyclomatique est supérieure à 19.50 et que la mesure n2 est inférieure à 132, il sera dès lors difficile de maintenir le logiciel ».

Certains attributs qui figurent au niveau des règles générées par CN2 ont été sélectionnés dans le choix des attributs pertinents (comme par exemple l'attribut 'exécutable' et l'attribut 'cyclomatique complexity'), ce qui confirme que ce choix est bien fondé.

**-M5'**

Même si certains algorithmes d'apprentissage n'ont pas donné de bons résultats, il serait intéressant d'analyser certaines des connaissances générées par ces derniers (règles induites et arbres de décision).

Par exemple voici un extrait de l'arbre de décision produit par M5' :

Inline_comments <= 25.5:

| executable<=137.5 :LM3(10/0%)

| executable>137.5 :LM4(27/91.555%)

**Interprétation de l'arbre :**

Si l'attribut 'inline_comments' est inférieur ou égal à 25.5 alors si l'attribut 'exécutable' est inférieur ou égal à 137.5 alors c'est la règle Lm3 qui permet de prédire la maintenabilté. Sinon, c'est Lm4 qui permet de prédire la maintenance

Sachant que :

Lm3 :
maintenance=+1

Lm4 :
Maintenance =-0.0208*inline_comments+0.0019*blank_lines+1.08059

La règle 4 se base sur deux attributs 'inline_comments' et 'blank_lines'. Il faudra remplacer ces attributs par leurs valeurs adéquates pour prédire la maintenance.

Comme nous pouvons le constater, le premier test de l'arbre de décision de M5' se base sur l'attribut 'inline_comments'; ensuite selon la valeur de cet attribut, il teste les deux attributs n2 et 'executable'. On confirme donc encore une fois que 'inline _comments' joue un rôle important dans la prédiction de la maintenabilité.

**-J48**

Si on prend le résultat de l'arbre généré par J48, on obtient :

:f(164.0/75)

number of leaves :1

Comme nous pouvons le constater, l'arbre de décision généré par J48 a été réduit à sa racine. Le classificateur J48 a trouvé qu'il serait meilleur de toujours prédire la classe facile pour toutes les instances. Ce qui explique que le nombre d'instances mal classées soit de 75 ; ce nombre représente le nombre d'instances étiquetées par la classe difficile.

Donc, par rapport à J48, on ne peut déduire grand chose concernant les attributs qui influent sur la prédiction de la maintenance.

**-OC1**

OC1 nous donne comme résultat un sous arbre de décision à 1 nœud. Ce dernier contient l'équation de l'hyperplan séparant au mieux l'ensemble des exemples :

Root Hyperplane :Left=[47,41], Right=[31.29]

1.0 x[11]+-4.00000=0

Les x[i] correspondent à une variable prédictive. Dans notre cas, x[11] correspond à l'attribut 'inline_comments'. Ainsi, OC1 se base uniquement sur le 11ème attribut qu'il juge pertinent pour la prédiction. Ce qui est conforme avec les résultats précédents.

Après avoir déterminé le meilleur algorithme permettant de prédire la maintenance, nous avons pu analyser certains résultats des autres algorithmes utilisés dans cette expérience. Nous avons trouvé qu'OC1, CN2 et Weka ont été en accord sur le fait que l'attribut prédominant permettant de prédire la maintenance était 'inline_comments'.

### 5.4.2 Comparaison avec d'autres résultats

L'expérience que nous avons menée a aussi été réalisée par Almeida et Al [2]. Dans le tableau suivant, nous allons donner le score obtenu par les algorithmes d'apprentissage utilisés par ces auteurs ainsi que nos résultats (Voir TAB. 5.5). Pour ne pas porter à confusion avec l'ancienne version de CN2, nous désignons par CN2' la version améliorée de CN2 que nous avons utilisé dans notre expérience.

| Modèle | Résultat |
|---|---|
| NewID | 48% |
| CN2 | 51% |
| C4.5 tree | 73% |
| CN2'(cross validation) | **73.33%** |
| Foil | 82% |
| CN2'(split) | **83.5%** |

**TAB. 5.5: Comparaison avec d'autres algorithmes d'apprentissage (Hypothèse H1)**

Si nous comparons nos résultats avec ceux obtenus par ces auteurs nous pouvons conclure que :

Par rapport à la cross-validation, nous avons obtenu un résultat aux alentours de 74%. Il s'avère un très bon résultat comparé à NewId et à l'ancienne version de CN2. Néanmoins, ce résultat n'est pas aussi bon que celui de FOIL qui détient un score de 82%. Par contre, le résultat obtenu par Split est de 86.5%. Cela signifie que nous avons amélioré les résultats obtenus avec une intervalle de 4.5%.

Il est vrai que la cross-validation est considérée comme le moyen le plus fiable pour valider un algorithme d'apprentissage mais la technique du Split est aussi valide.

Nous avons donc prouvé que l'emploi de la version améliorée de CN2 permet de prédire la maintenance corrective grâce aux mesures utilisées dans l'expérience. Notons que le meilleur attribut pour la prédiction est 'inline_comments', sans oublier les autres attributs sélectionnés dans le Tableau 5.3.

## 5.5 Résultats expérimentation 2

### 5.5.1 Hypothèse H2a

Nous allons commencer par valider l'hypothèse H2a qui spécifie que:

*La position d'un composant dans une hiérarchie de classes du système affecte d'une certaine manière sa réutilisabilité.*

#### 5.5.1.1 Résultats et interprétations

**Résultat de la sélection d'attributs :**

**Attribute Evaluator** : Ranked
**Search méthode** : BestFirst
**Liste déroulant** : Prédiction de la variable Reusability.

| Poids | Attributs |
|---|---|
| 50 | NOC |
| 50 | HIT |
| 50 | NOD |
| 50 | NOP |
| 50 | OIM |
| 41.667 | DIT |
| 41.667 | NOA |

**TAB. 5.7: Résultat de la sélection d'attributs pour l'hypothèse H2a**

Le tableau 5.7, nous donne la liste des attributs qui jouent un rôle prédominant dans la prédiction. Nous allons essayer de voir si ce résultat concorde avec le choix des algorithmes d'apprentissage.

**Taux de succès**

Après avoir effectué plusieurs essais, nous avons résumé les résultats obtenus dans le tableau suivant:

| | Train | Cross_validation (**K**) | Split(**%**) |
|---|---|---|---|
| OC1 | 75% | 48.80%(10) | 42.40%(71) |
| CN2 | 84.17% | 67.31%(4) | 71.5%(78) |
| J48 | 70.23% | 53.57%(19) | 55.17%(66) |
| OneR | 52.38% | 45.23%(9) | 52.63%(78) |
| Ibk | 95.23% | 54.76%(10) | 53.57%(67) |
| Neural Network | 58.33% | 53.57%(7) | 60.71%(76) |
| SVM | 75% | 45.23%(5) | 52.5%(79) |
| Part | 80.95% | 55.95%(84) | 69%(73) |
| M5' | 60.02% | 48.64%(7) | 47.25%(76) |
| Naïve bayes | 76.19% | 54.76%(9) | 65.51%(66) |

**TAB. 5.8: Résultats influence de l'héritage sur la réutilisabilité.**

Les images suivantes présentent les résultats du tableau 5.10 sous forme d'histogrammes :

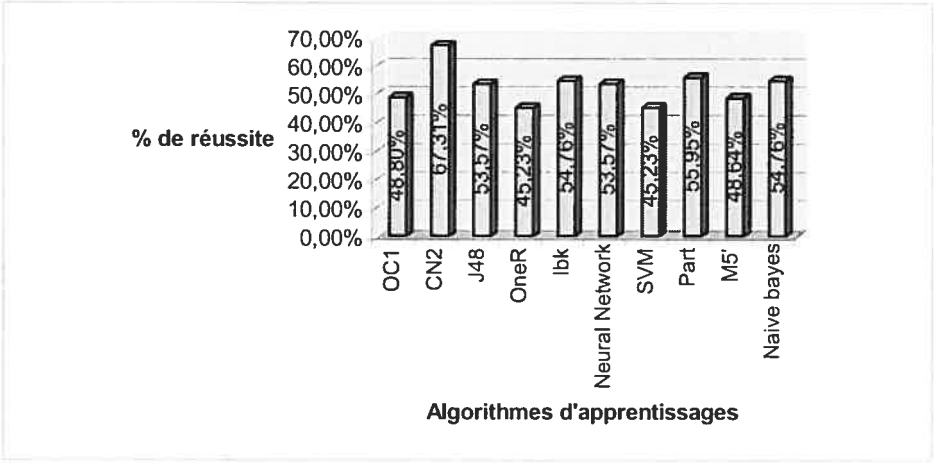

**FIG. 5.5: Histogramme hypothèse H2a(cross-validation)**

Comme nous pouvons le remarquer, le meilleur résultat obtenu est de 67.31% pour la cross validation et 71.5% pour le Split. C'est l'algorithme d'apprentissage CN2 qui donne le meilleur score. Néanmoins, on remarque que Part donne aussi un bon résultat pour le Split.

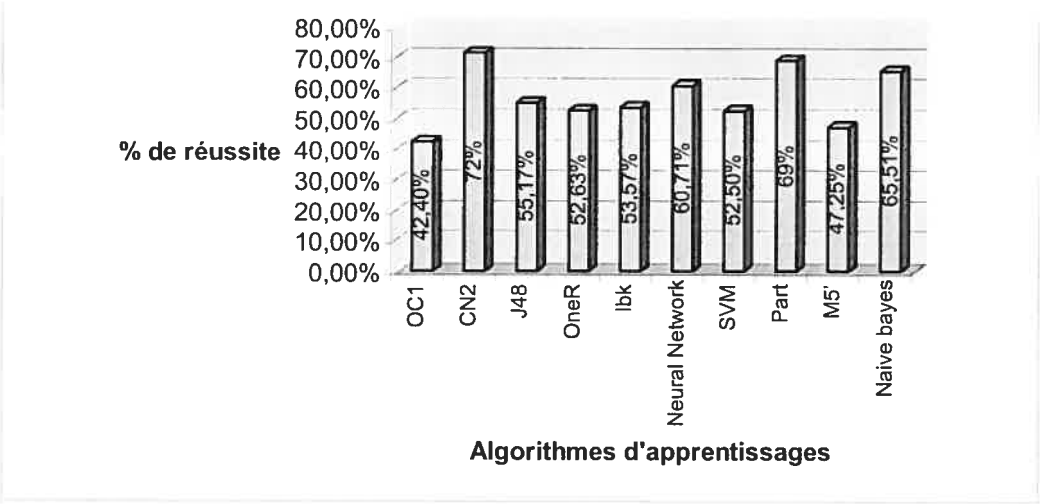


**FIG. 5.6 : Histogramme hypothèse H2a(Split)**

Pour aboutir à ce résultat, nous avons effectué plusieurs expériences. Voici les paramètres d'apprentissage de CN2 et Part qui ont donné les meilleurs résultats:

Paramètres d'apprentissage de CN2 :

Fonction d'estimation d'erreurs : Laplace.

Type de règles générées: Non ordonnées.

Seuil de signification : 75%

Paramètres d'apprentissage de Part :

Facteur de confidence : 25%

Nombre minimum d'exemples par feuilles :4

Voici quelques règles générées par Cn2 :

| | | |
|---|---|---|
| Règle1 :<br><br>Si **OIM** < 16.00<br>et **NOP** > 6.50<br><br>→ Reusability '1' | Règle 2:<br><br>Si **NOC** < 0.50<br>et **OIM** > 28.00<br>et **NOP** > 7.50<br>→ Reusability '1' | Règle 3:<br><br>Si 24.50 < **OIM** < 30.50<br>et **NOP** < 7.50<br>→ Reusability '2' |
| Règle 4:<br><br>Si **DIT** < 0.50<br>et **OIM** < 4.00<br>→ Reusability '2' | Règle 5:<br>Si **NOD** > 1.50<br>et **OIM** < 22.00<br>et 50<**NOP**< 4.50<br>→ Reusability '3' | |

**TAB. 5.9: Règles générées par CN2(hypothèse H2a)**

Ces règles nous aident à prédire le degré de réutilisabilité selon certaines valeurs de mesures internes concernant l'héritage par exemple, la règle 2 indique :

« La classe Reuse est totalement réutilisable si la mesure d'héritage NOC est inférieure à 0.50, la mesure OIM est supérieur à 28 et que la mesure NOP est supérieur à 7.50 »

Pour avoir une idée sur les autres types de connaissances générées, voici un extrait de l'arbre de décision produit par l'algorithme J48 :

**NOP** <= 4
| **HIT** <= 0: 4 (47.0/15.0)
| HIT > 0
| | **OIM** <= 21: 3 (7.0/1.0)
| | OIM > 21: 2 (2.0/1.0)
NOP > 4
| DIT <= 2
| | NOP <= 10
| | | NOP <= 5
| | | | NOC <= 2
| | | | | OIM <= 14: 3 (6.0/1.0)
| | | | | OIM > 14: 1 (2.0/1.0)
| | | | NOC > 2: 2 (2.0)
| | | NOP > 5: 2 (14.0/6.0)
| | NOP > 10: 4 (2.0)
| DIT > 2: 4 (2.0)

On remarque bien que J48 base essentiellement sa première condition sur l'attribut NOP ensuite HIT et DIT. Tous ces attributs ont déjà été déclarés comme attributs pertinents dans la prédiction de la réutilisabilité, ce qui renforce encore une fois la validité des résultats obtenus.

-Si on consulte les règles pour OneR, on a :

**HIT**:
< 0.5 -> 3
< 1.5 -> 2
>= 1.5 -> 1

OneR choisit HIT comme attribut principal. Ce dernier fait partie des attributs choisis par la sélection d'attributs pertinents.

Nous pouvons d'ores et déjà affirmer que certains attributs internes tels que NOC, HIT, NOD, NOP et OIM jouent un rôle important pour prédire la réutilisation. Le meilleur modèle de prédiction que nous pouvons juger

important serait d'utiliser la version améliorée de CN2 . Part aussi serait recommandé pour la prédiction.

Et même si certains algorithmes ne donnent pas de très bons résultats pour le taux de réussite, nous pouvons aussi les exploiter pour valider les tests.

### 5.5.2 Hypothèse H2b

Dans ce qui suit, nous allons essayer de valider l'hypothèse H2b :

'Influence du Couplage (niveau conception) sur la réutilisabilité '.

#### 5.5.2.1 Résultats et interprétations

**Résultat de la sélection d'attributs :**

**Attribute Evaluator** : Ranked

**Search méthode** : BestFirst

**Liste déroulant** : Prédiction de la variable reusability.

| Poids | Attributs |
|---|---|
| 0.062801 | ACMIC |
| 0.048005 | CBO |
| 0.047247 | OCMEC |
| 0.037173 | OMMEC |
| 0.025014 | OCAEC |
| 0.017427 | NOC |
| 0.01284 | DCMEC |
| 0.012614 | AMMEC |
| 0.011187 | WMC |

**TAB. 5.10: Résultat de la sélection d'attributs pour l'hypothèse H2b**

Le tableau 5.10 nous donne un aperçu sur le résultat de la sélection d'attributs pertinents.

**Taux de succès**

Le T 5.13 résume les résultats obtenus grâce aux algorithmes d'apprentissage retenus pour cette hypothèse :

| | Train | Cross_validation(**K**) | Split(%) |
|---|---|---|---|
| OC1 | 37.5% | 51.19%(6) | 48.8%(75) |
| CN2 | 93.4% | 62.5%(5) | 73.5%(77) |
| J48 | 88.09% | 60.71%(7) | 71%(77) |
| OneR | 57.14% | 53.57%(17) | 53.84%(70) |
| Ibk | 100% | 66.66%(9) | 68.96%(66) |
| Neural Network | 85.71% | 64.28%(84) | 66.66%(68) |
| SVM | 98.80% | 61.90%(9) | 73.5%(75) |
| Part | 89.28% | 61.90%(18) | 68.96%(66) |
| M5' | 59.02% | 50.25%(19) | 52.12%(76) |
| Naive bayes | 90.47% | 63.09%(7) | 71.42%(92) |

**TAB. 5.11: Influence du Couplage (niveau conception)**

**sur la réutilisabilité**

Les graphiques suivants présentent les résultats obtenus sous forme d'histogrammes :

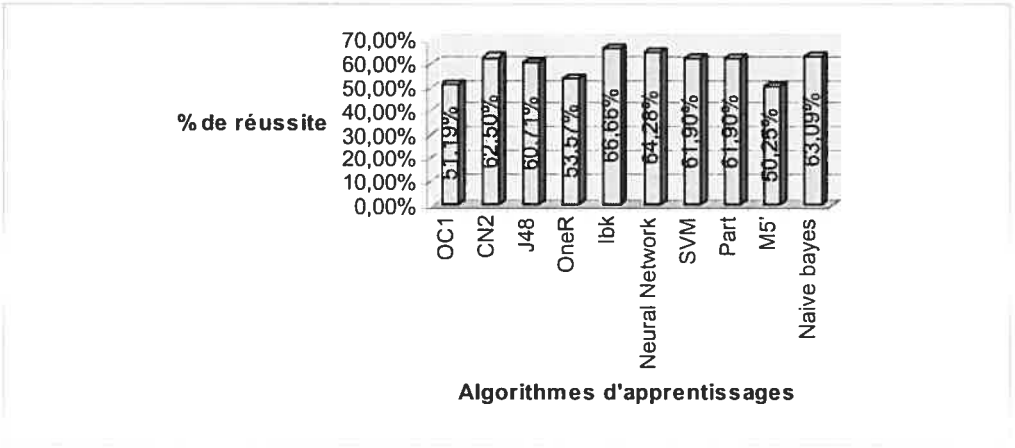


**FIG. 5.7: Histogramme hypothèse H2b (cross-validation)**

De ces résultats, nous pouvons constater que le couplage affecte la réutilisabilité sur un degré de confiance de 73.5% avec la technique du Split en utilisant la version améliorée de CN2 et avec une confiance de 66.66% avec la technique de la cross validation en utilisant IBK .

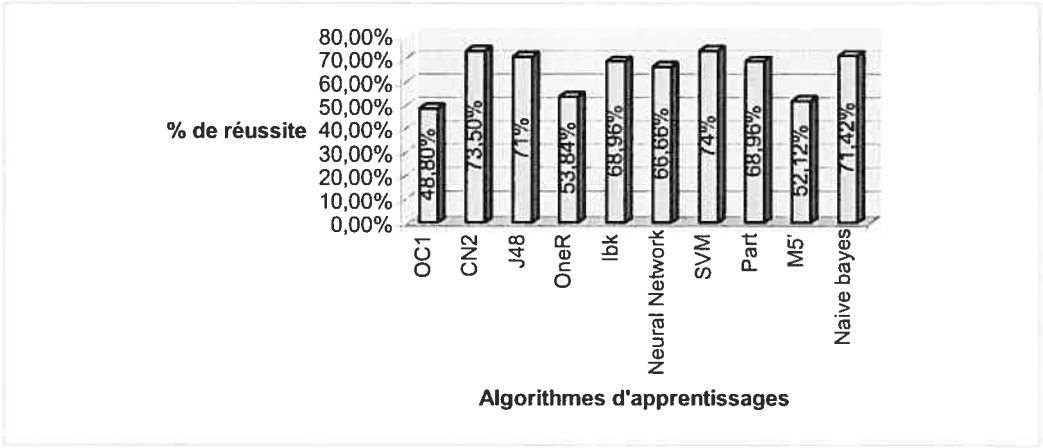


**FIG 5.8: Histogramme hypothèse H2b(Split)**

Paramètres d'apprentissage de CN2 :

Fonction d'estimation d'erreurs : Laplace.

Type de règles générées: Non ordonnées.

Seuil de signification : 65%

Autre paramètres : par défaut.

Paramètres d'apprentissage de Neural Network :

Apprentissage :par Backpropagation

Hiddenlayers : 4

a :0.4

Autres paramètres : par défaut

Même si le réseau de neurones (Neural Network) montre de meilleurs résultats par rapport à la cross-validation, cette méthode ne génère pas de connaissance. Voici donc quelques règles générées par CN2 :

| | | |
|---|---|---|
| Règle 1:<br>Si 42.00<**OCMEC** < 51.00<br>→ Reusability '1' | Règle 2 :<br>Si **CBO** < 2.00<br>→ Reusability '1' | Règle3 :<br>Si **CBO** > 15.50<br>et **OCMIC** < 15.50<br>et **OMMIC** > 2.00<br>→ Reusability '2' |
| Règle 4:<br>Si **CBO** > 4.50<br>Et 3.50 < **OCMIC** < 5.50<br>→ Reusability '3' | | |

**TAB. 5.12: Règles générées par Cn2(hypothèse H2b)**

En considérant la règle 4, nous pouvons dire :

« Une classe est réutilisable avec un effort important si la mesure de couplage CBO est supérieure à 4.50 et que la mesure OCMIC est totalement comprise entre 3.5 et 5.50 »

- Résultats obtenus par OC1

Root Hyperplane: Left = [1,0,7,10], Right = [4,13,13,28]

1.000000 **x[23]** + -2.500000 = 0

Comme nous pouvons le constater, OC1 attribue le plus grand poids au 23 ème attribut; ce dernier correspond à OMMEC.

-Voici quelques règles générées par OneR

**OMMEC**:

< 1.5 -> 4

< 3.5 -> 3

< 63.5 -> 4

>= 63.5 -> 2

OneR renforce le choix de OC1 ainsi que le choix de la sélection d'attributs. Ce dernier se base lui aussi sur l'attribut OMMEC pour prédire la variable Reusability.

L'algorithme d'apprentissage CN2 se base sur les attributs OCMEC, CBO, OCMIC et OMMIC afin de faire la prédiction. Certains de ces attributs figurent au niveau de la liste des attributs pertinents. Néanmoins, il ne faut pas ignorer que même si certains algorithmes d'apprentissage comme OneR et OC1 n'ont pas donné un bon taux de réussite escompté, ils ont eu comme point en commun de choisir l'attribut OMMEC pour la prédiction.

Nous pouvons donc conclure que les mesures de couplage sélectionnées peuvent prédire la réutilisabilité avec un certain taux de confiance .

### 5.5.3 Hypothèse H2c

*Influence du Couplage (niveau code) sur la réutilisabilité.*

#### 5.5.3.1 Résultats et interprétations

**Résultat de la sélection d'attributs :**

**Attribute Evaluator** : Ranked

**Search méthode** : BestFirst

**Liste déroulant** : Prédiction de la variable Reusability

| Poids | Attributs |
|---|---|
| 0.545 | TYIexp |
| 0.368 | TYI |
| 0.313 | NPIexp |
| 0.303 | RCI |
| 0.296 | SRCIimp |
| 0.282 | SRDIimp |
| 0.276 | RDIexp |
| 0.274 | RDI |
| 0.267 | RCIexp |
| 0.248 | NPI |

**TAB. 5.13: Résultat de la sélection d'attributs pour l'hypothèse H2c**

Dans le tableau 5.13, on retrouve l'ensemble des attributs jugés pertinents pour la prédiction.

**Taux de succès**

Voici les résultats obtenus avec les algorithmes d'apprentissage :

| | Train | Cross_validation(K) | Split(%) |
|---|---|---|---|
| OC1 | 25% | 53.57%(5) | 42.80%(75) |
| CN2 | 95.8% | 63.1%(8) | 70.8%(71) |
| J48 | 90.47% | 65.47%(27) | 69%(70) |
| OneR | 61.90% | 55.95%(7) | 60%(72) |
| Ibk | 100% | 63.09%(14) | 80.22%(74) |
| Neural Network | 88.09% | 57.14%(73) | 62.06%(66) |
| SVM | 98.90% | 59.52%(3) | 73.25%(78) |
| Part | 91.66% | 60.71%(23) | 71%(77) |
| M5' | 45.20% | 51.19%(13) | 53.5%(75) |
| Naive bayes | 86.90% | 66.66%(7) | 74.77%(79) |

**TAB. 5.14: Résultats couplage (niveau code) versus réutilisabilité**

Les graphiques 5.9 et 5.10 présentent les résultats obtenus sous forme d'histogrammes :

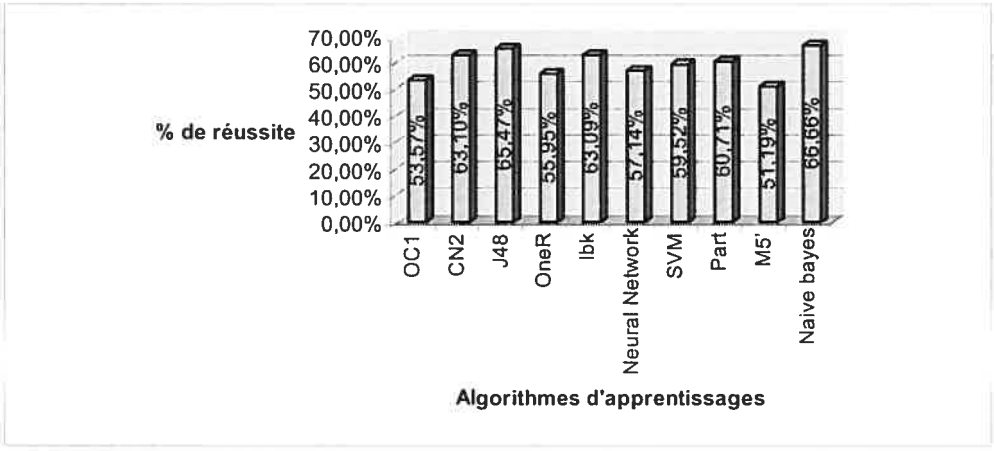


**FIG. 5.9 : Histogramme hypothèse H2c(cross-validation)**

Nous pouvons constater que Naïve Bayes donne un résultat de 66.66% avec une cross- validation égale à 7. À priori, cela ne constitue guère un bon résultat, néanmoins nous avons pu atteindre un taux de réussite de 80.22%, avec un Split égal à 74% grâce à l'algorithme d'apprentissage Ibk.

Pour l'apprentissage de Ibk, nous avons donné plusieurs valeurs au paramètres K (K plus proches voisins), et la valeur 1 a donné le meilleur résultat.

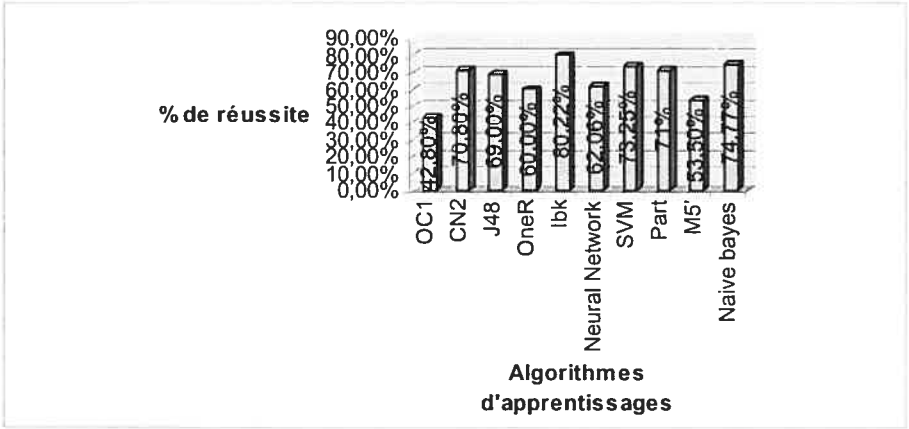


**FIG. 5.10 : Histogramme hypothèse H2c (Split)**

Voici un extrait de l'arbre de décision généré par J48 :

**TYIexp** <= 5
| SRDIimp <= 1
| | SRDIexp <= 0
| | | **RCI** <= 0: 3 (20.0/5.0)
| | | RCI > 0
| | | | RDI <= 17
| | | | | NPIexp <= 2
| | | | | | StRDIimp <= 1: 4 (21.0)
| | | | | | StRDIimp > 1
| | | | | | | RCIexp <= 0: 3 (2.0)
| | | | | | | RCIexp > 0: 4 (5.0)
| | | | | **NPIexp** > 2
| | | | | | NPI <= 14: 3 (2.0)
| | | | | | NPI > 14: 4 (2.0)

Au niveau de l'arbre de J48, on a commencé par tester l'attribut TYIexp en premier. Cela nous permet de confirmer que c'est un attribut pertinent pour la prédiction. Nous retrouvons aussi dans l'arbre d'autres attributs qui ont été sélectionnés au niveau de la liste du tableau 5.13 (comme RCI et NPIexp)

**-Extrait des règles générées par OneR :**

**TYIexp**:
< 4.5 -> 4
< 36.5 -> 1
>= 36.5 -> 2

Avec OneR, on confirme encore une fois que TYIexp est un attribut pertinent.

A travers cette expérience, on peut donc affirmer que les mesures sélectionnées peuvent prédire la variable Reusability; le couplage au niveau code affecte donc la réutilisabilité avec un degré de confiance de 83.33% avec le Split et avec un degré de confiance de 63.09% avec la cross-validation.

### 5.5.4 Hypothèse H2d

*influence de la complexité sur la réutilisabilité .*

#### 5.5.4.1 Résultats et interprétations

**Résultat de la sélection d'attributs**

**Attribute Evaluator** : Ranked

**Search méthode** : BestFirst

**Liste déroulant** : Prédiction de la variable Reusability

| **Poids** | **Attributs** |
|---|---|
| 0.226 | **NPM** |
| 0.208 | **CSB** |

**TAB. 5.15: Résultat de la sélection d'attributs pour l'hypothèse H2d**

On remarque qu'il y'a seulement deux attributs que la sélection a jugé pertinents pour la prédiction.

**Taux de succès**

| | Train | Cross_validation(**K**) | Split(%) |
|---|---|---|---|
| OC1 | 50% | 54.76%(6) | 52.38%(76) |
| CN2 | 100% | 60%(5) | 69.3%(74) |
| J48 | 94.04% | 59.52%(16) | 66.5%(72) |
| OneR | 54.76% | 42.85%(5) | 52.38%(75) |
| IBk | 100% | 60.71%(14) | 72(79) |
| Neural Network | 78.57% | 65.47% (5) | 83.66%(78) |
| SVM | 98.80% | 54.76%(9) | 78.5%(76) |
| Part | 95.23% | 58.33%(16) | 75.5%(78) |
| M5' | 16.25% | 30.75%(11) | 21.42%(56) |
| Naibe bayes | 50% | 45.23%(7) | 37%(79) |

**TAB. 5.16: Influence de la complexité sur la réutilisabilité**

Les graphiques suivants présentent les résultats obtenus sous forme d'histogrammes :

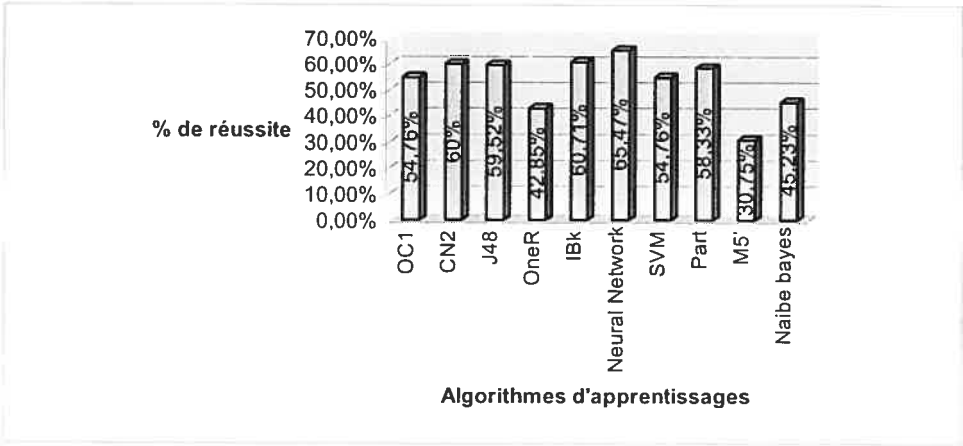


**FIG. 5.11 : Histogramme hypothèse H2d(Cross-validation)**

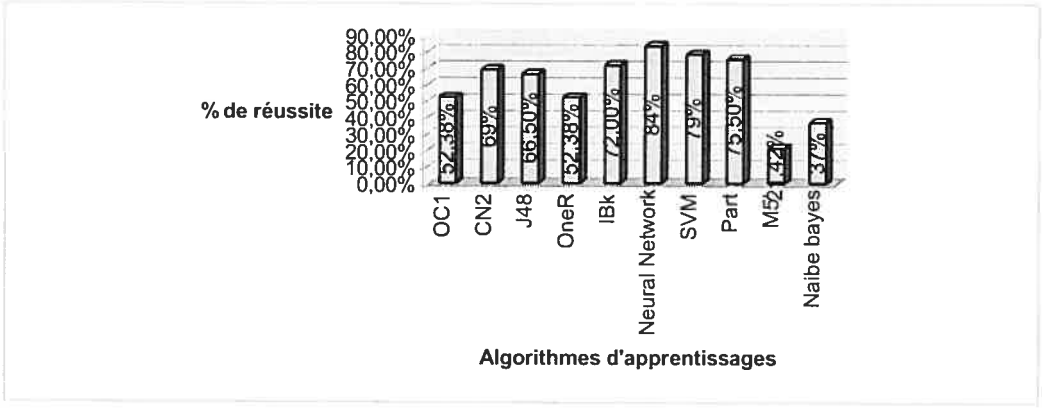


**FIG. 5.12 : Histogramme hypothèse H2d(Split)**

Comme nous pouvons le constater, avec le réseau de neurones de Weka, nous avons pu atteindre un taux de réussite de 65.47%, avec la cross-validation. Le meilleur score a atteint 83.66% avec un Split égal à 78%.

Le réseau de neurones étant incapable de produire de la connaissance (règles et arbres de décisions), voici donc quelques règles générées par Cn2 :

| Règle3 : | Règle 2: | Règle 3: | Règle 4 : |
|---|---|---|---|
| Si NOM < 30.50<br>et CIS > 12.50<br>→ Reusability '1' | Si RFC > 11.50<br>→ Reusability '2' | Si NPA < 0.50<br>et 10.00<**CSB**< 512.00<br>→ Reusability '3' | Si CIS < 24.50<br>et **NPM** > 1.04<br>et **CSB** > 8.00<br>→ Reusability '4' |

**TAB. 5.17: Règles générées par Cn2(hypothèse H2d)**

Si nous essayons de traduire la règle 4 , cela signifie que :
« Un composant n'est pas réutilisable, si sa mesure de complexité CIS est inférieure à 24.50, si sa mesure NPM est supérieure à 1.04 et si sa mesure de complexité CSB est supérieure à 8.00 »

Voici une portion de l'arbre généré par J48 :

J48 pruned tree
---

**CSB** <= 12
| NOP <= 4
| | NOM <= 11
| | | NOM <= 9: 2 (2.0)
| | | NOM > 9: 3 (2.0)
| | NOM > 11: 4 (5.0/1.0)

Comme nous pouvons le constater, le premier test de l'arbre se base sur l'attribut CSB qui a été élu parmi les attribut prépondérants dans la prédiction. Nous pouvons faire la même remarque aussi sur l'arbre de OC1 qui se base sur le 12éme attribut qui est CSB:

Root Hyperplane: Left = [0,0,1,0], Right = [0,1,0,0]
1.0 **x[12]** + -80.000000 = 0

Avec cette expérience, nous pouvons affirmer que les mesures sélectionnées peuvent prédire la variable Reusability; la complexité affecte la réutilisabilité avec un degré de confiance qui peut atteindre 90% avec le Split et 65.47% avec la cross-validation.

### 5.5.5 Comparaison avec d'autres résultats

Nous allons comparer nos résultats aux résultats de l'étude obtenus par Y. Mao [4]. Cette étude a été faite sur les mêmes données que notre expérience en exploitant l'algorithme C4.5 (Voir TAB. 5.18).

| **Hypothèse** | H2a | | H2b | | H2c | | H2d | |
|---|---|---|---|---|---|---|---|---|
| | **CV**[1] | **Split** | **CV** | **Split** | **CV** | **Split** | **CV** | **Split** |
| C4.5 | **73.8%** | | **88.1%** | | **86.9%** | | **89.3%** | |
| CN2 | 67% | 71.5 % | 62.5% | 73.5% | | | | |
| Naïve-bayes | | | | | 67% | 75% | | |
| RN | | | 64% | 67% | | | 66% | 84% |
| IBK | | | | | 63% | 80.2% | | |

**TAB. 5.18: Comparaison de nos résultats avec celle de C4.5**

[1] Cross Validation

Les résultats obtenus par C4.5 sont mieux que ceux obtenus par nos algorithmes. Mais sans certains cas, nous nous sommes rapprochés des résultats obtenus par C4.5. Cependant dans notre étude, nous avons désigné l'ensemble des attributs qui sont jugés importants pour la prédiction de la réutilisabilité.

Pour les quatres hypothèses énoncées : héritage, couplage au niveau conception, couplage au niveau code et complexité, on a obtenu des résultats qui varient entre 62.5% et 67% avec la technique de la cross-validation , et des résultats variant entre 67% et 84% avec le Split.

## 5.6 Résultats expérimentation 3

Nous allons essayer de valider les hypothèses H3a, H3b ainsi que H3c présentées dans le chapitre 4 :

### 5.6.1 Hypothèse H3a

*Influence des mesures d'héritage sur la propension à engendrer des erreurs.*

#### 5.6.1.1 Résultats et interprétations

**Résultat de la sélection d'attributs :**

**Attribute Evaluator** : Ranked
**Search méthode** : BestFirst
**Liste déroulant** : Prédiction de la variable erreur.

| Poids | Attributs |
|---|---|
| 0.04429 | **NMI** |
| 0.03436 | **NMA** |
| 0.01829 | NOD |

| | |
|---|---|
| 0.01829 | NOD |
| 0.01416 | CLD |
| 0.00985 | NOC |

**TAB. 5.19: Résultat de la sélection d'attributs pour l'hypothèse H3a**

Comme nous pouvons le constater, 5 attributs ont été sélectionnés. Les attributs prépondérants dans la prédiction sont respectivement NMI, NMA, NOD, CLD et NOC.

**Taux de succès**

Le tableau suivant illustre les résultats obtenus par les algorithmes d'apprentissage :

| | Train | Cross_validation(K) | Split(%) |
|---|---|---|---|
| OC1 | 100% | 72.5%(80) | 70%(50) |
| CN2 | 88% | 87.5%(8) | 89%(77) |
| J48 | 85% | 75%(3) | 81.48%(67) |
| OneR | 71.25% | 72.5%(3) | 80%(69) |
| Ibk | 96.25% | 73.75%(17) | 77.77%(67) |
| Neural Network | 16.25% | 33.75%(5) | 87%(78) |
| SVM | 15% | 38.75%(5) | 43.45%(76) |
| Part | 85% | 80%(30) | 85.18%(67) |
| M5' | 58.33% | 47.61%(28) | 57.14%(63) |
| Naive bayes | 21.25% | 43.75%(16) | 30%(78) |

**TAB. 5.20: Résultats influence de l'héritage sur la propension à engendrer des erreurs.**

Concernant la cross-validation, nous obtenons l'histogramme suivant :

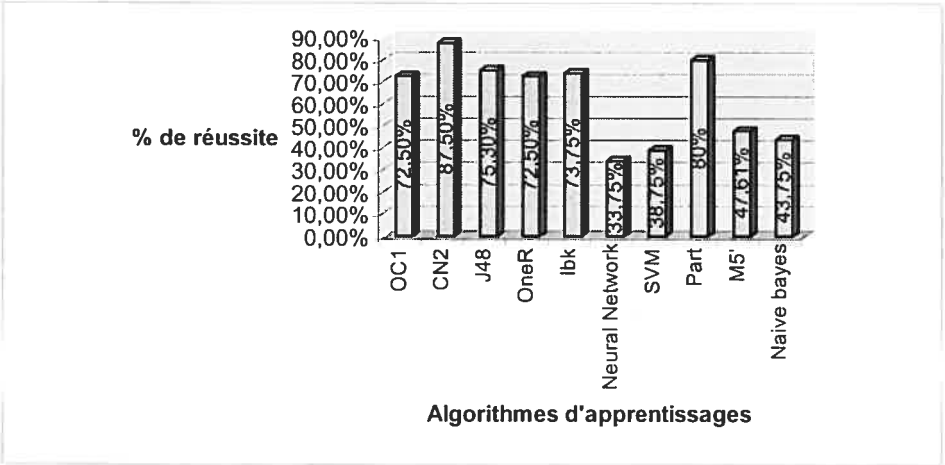


**FIG. 5.13: Histogramme hypothèse H3a (cross-validation)**

Comme nous pouvons le remarquer au niveau du tableau 5.20 et encore mieux au niveau des histogrammes, CN2 nous offre le meilleur résultat. Nous avons obtenu un taux de réussite de 87.5% avec la cross-validation (voir Fig 5.15 et 5.16) et un taux de succès de 89% avec le Split; on peut donc conclure qu'il existe bien un lien causal entre les mesures sélectionnées et la variable à prédire.

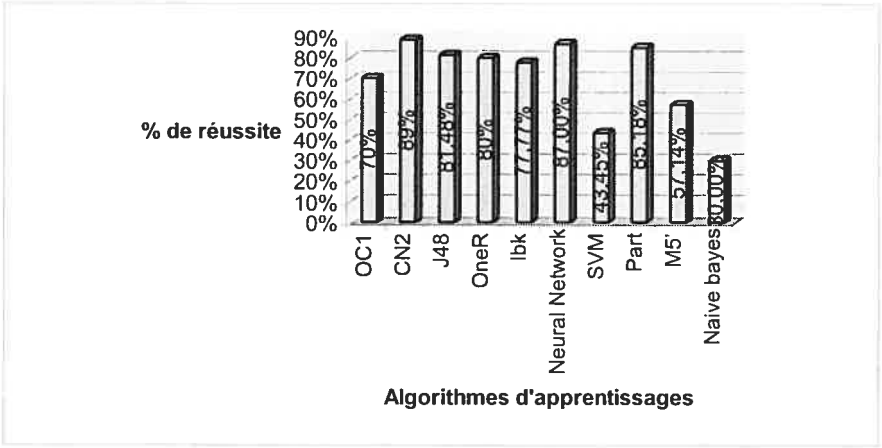


**FIG 5.14: Histogramme hypothèse H3a (Split)**

Pour l'apprentissage de CN2, nous avons utilisé les paramètres suivants :

Fonction d'estimation d'erreurs : Laplace.

Type de règles générées: Non ordonnées.

Seuil de signification : 89%

Voici quelques règles générées par CN2 :

| | | |
|---|---|---|
| Règle1 :<br><br>Si **NMI** < 18.50<br>et 13.50 < **NMA**< 20.00<br>→ Erreur '1' | Règle 2:<br><br>Si NOD > 2.50<br>et **NMA** > 19.00<br>→ Erreur '1' | Règle 3:<br><br>Si DIT < 2.00<br>et 5.50 < NMI < 25.00<br>et **NMA** < 11.50<br>→ Erreur '1' |
| Règle 4:<br><br>Si NMI > 25.00<br>→ Erreur '2' | (DEFAULT) :<br>→ Erreur '1' | |

**TAB. 5.21: Règles générées par CN2 (hypothèse H3a)**

Si on essaie de traduire La règle 1, on peut affirmer que :

« Si la mesure d'héritage NMI est inférieure à 18.50 et que la mesure NMA est comprise entre 13.50 et 20; dans ce cas, il y'a de faibles chances pour que le composant contienne des erreurs »

-Voici une portion d'arbre générée par J48

**NMI** <= 22
| DIT <= 1
| | NOD <= 8
| | | NMO <= 8
| | | | NMO <= 6
| | | | | CLD <= 0: 1 (36.0/8.0)
| | | | | CLD > 0

Comme dans le cas de la sélection d'attribut, J48 confirme que l'attribut NMI est dominant.

**-OneR**

**CLD**:

< 2.5 -> 2
>= 2.5 -> 1

OneR choisit CLD comme attribut principal. On voit apparaître cet attribut au niveau de l'arbre générée par J48.

### 5.6.2 Hypothèse H3b

*Influence des mesures de Cohésion sur la propension à engendrer des erreurs .*

#### 5.6.2.1 Résultats et interprétations

**Résultat de la sélection d'attributs :**

**Attribute Evaluator** : Ranked

**Search méthode** : BestFirst

**Liste déroulant** : Prédiction de la variable Erreur

| Poids | Attributs |
|---|---|
| 69.62 | Coh |
| 69.62 | LCOM1 |
| 69.62 | ICH |
| 68.354 | LCOM5 |
| 65.823 | LCOM2 |
| 65.823 | TCC |
| 65.823 | Co |
| 64.557 | LCOM3 |
| 64.557 | LCC |
| 64.557 | LCOM4 |

**TAB. 5.22: Résultat de la sélection d'attributs pour l'hypothèse H3b**

Dans le tableau 5.22, on retrouve les attributs classés selon l'ordre de pertinence.

**Taux de succès**

| | Train | Cross_validation(K) | Split(%) |
|---|---|---|---|
| OC1 | 57.14% | 70.88%(79) | 68.35%(75) |
| CN2 | 88.76% | 72.22%(8) | 74.8%(78) |
| J48 | 73.41 % | 69.62 %(15) | 80.25%(78) |
| OneR | 78.48% | 65.82 %(14) | 73%(76) |
| IBK | 97.46% | 70.88%(19) | 92%(76) |
| Neural Network | 75.94% | 68.35%(17) | 85%(75) |
| SVM | 70.88% | 70.88%(4) | 81%(73) |
| Part | 73.41% | 69.62%(20) | 85%(76) |
| M5' | 54.29% | 48.22%(6) | 82%(71) |
| Naïve bayes | 94.93% | 77.21%(4) | 86.77%(75) |

**TAB. 5.23: Résultats influence de la cohésion sur la propension à engendrer des erreurs.**

Voici les Résultat sous forme d'histogrammes :

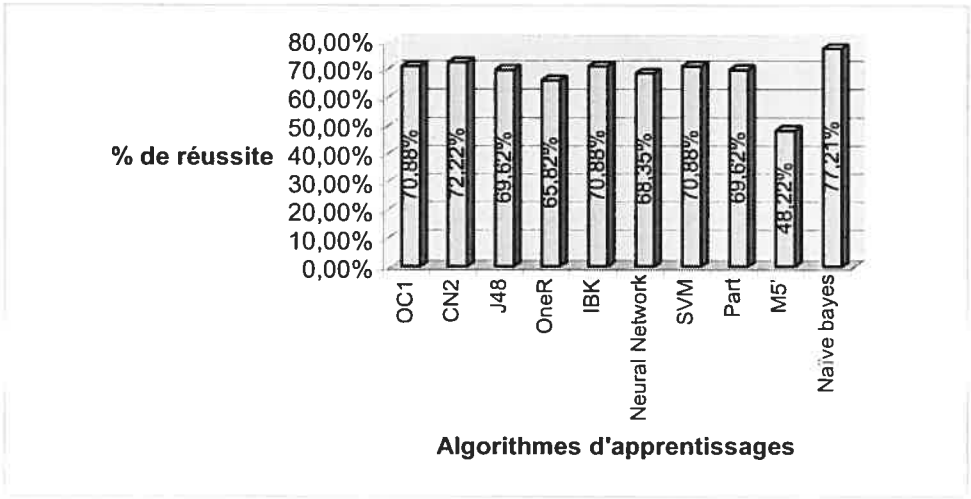


**FIG. 5.15: Histogramme hypothèse H3b (cross-validation)**

Nous remarquons que Naïve Bayes offre le meilleur résultat, 77.21% pour la cross- validation et un score de 86.77% pour le Split., ce qui nous permet de renforcer l'hypothèse énoncée.

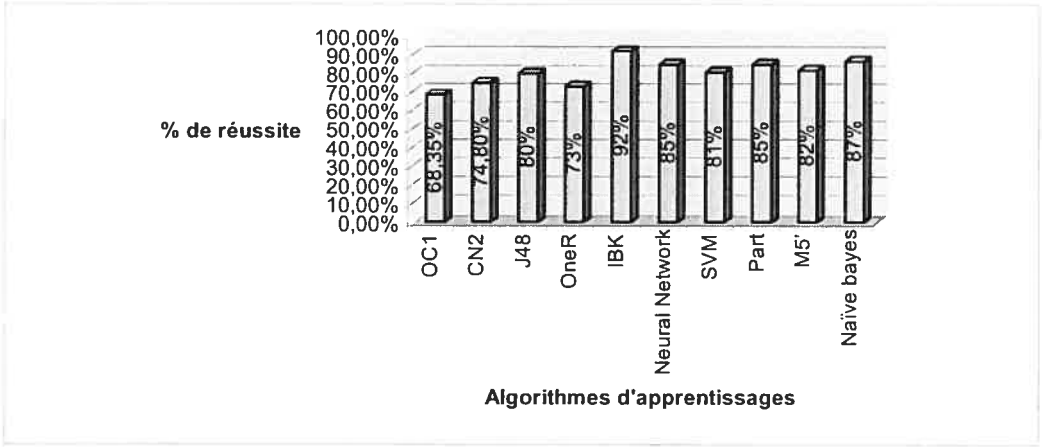


**FIG. 5.16: Histogramme hypothèse H3b (Split)**

Voici quelques règles générées par CN2 :

| | | |
|---|---|---|
| Règle1 :<br><br>Si **LCOM1** > 5.00<br>et LCOM4 < 3.50<br>et LCOM5 > 0.13<br>→ Erreur '1' | Règle 2:<br><br>Si LCOM2 < 8.50<br>et Coh < 0.22<br><br>→ Erreur '1' | Règle 3:<br>Si LCOM4 < 32.00<br>et LCOM5 < 0.70<br>et LCC < 0.47<br>et TCC > 0.19<br>→ Erreur '1' |
| Règle 4:<br><br>Si 3.50 < **LCOM1** < 5.00<br>→ Erreur '2' | Règle 5:<br><br>Si **LCOM1** < 176.50<br>et LCOM4 > 99.50<br>→ Erreur '2' | |

**TAB. 5.24: Règles générées par CN2 (hypothèse H3b)**

CN2 se base sur presque tous les attributs mais on voit bien que LCOM1 prédomine. Ceci justifie que dans la sélection d'attributs (Ranked), LCOM1 ait obtenu un grand poids par rapport aux autres.

Si on consulte la règle 4 on déduit que :
« Le composant a un faible risque de contenir des erreurs si la mesure de cohésion LCOM1 est strictement comprise entre 3.50 et 5 »

**-Arbre généré par J48**

**LCOM2** <= 10: m (72.0/19.0)
**LCOM2** > 10: b (7.0/2.0)

J48 se base sur l'attribut Lcom2 ce qui concorde avec la sélection d'attributs.
**-M5'**

**LM1** (79/107.622%)

le classificateur M5' a résolu qu'il serait meilleur de toujours prédire la classe LM1 pour toutes les instances.

### 5.6.3 Hypothèse H3c

*Influence des mesures de Couplage sur la propension à engendrer des erreurs.*

#### 5.6.3.1 Résultats et interprétations

**Résultat de la sélection d'attributs :**

**Attribute Evaluator** : Ranked

**Search méthode** : BestFirst

**Liste déroulant** : Prédiction de la variable Erreur

| Poids | Attributs |
|---|---|
| 0.032093 | IH-ICP |
| 0.029678 | AMMIC |
| 0.025195 | RFCoo |
| 0.023107 | RFC1 |
| 0.015423 | OCMIC |
| 0.014582 | DMMEC |
| 0.013079 | DCMEC |
| 0.012667 | CBO |

**TAB. 5.25: Résultat de la sélection d'attributs pour l'hypothèse H3c**

**Taux de succès**

Le tableau suivant illustre les résultats obtenus pour chaque algorithme d'apprentissage :

| | Train | Cross_validation(K) | Split(%) |
|---|---|---|---|
| OC1 | 100% | 73.75%(5) | 70%(75) |
| CN2 | 93.5% | 78.53%(26) | 58.22%(70) |
| J48 | 87.5% | 77.5%(19) | 88.5%(72) |
| OneR | 80 % | 76.25%(27) | 84%(77) |
| IBk | 100% | 71.25%(11) | 73%(76) |
| Neural Network | 85% | 72.5%(13) | 79.16%(70) |
| SVM | 72.5% | 82.5%(4) | 85.5%(67) |
| Part | 86.25% | 81.25%(35) | 87.77%(73) |
| M5' | 62.33% | 60.41%(10) | 67.77%(17) |
| Naïve bayes | 95% | 72.5%(12) | 79.5%(75) |

**TAB. 5.26: Résultats influence du couplage sur la propension à engendrer des erreurs.**

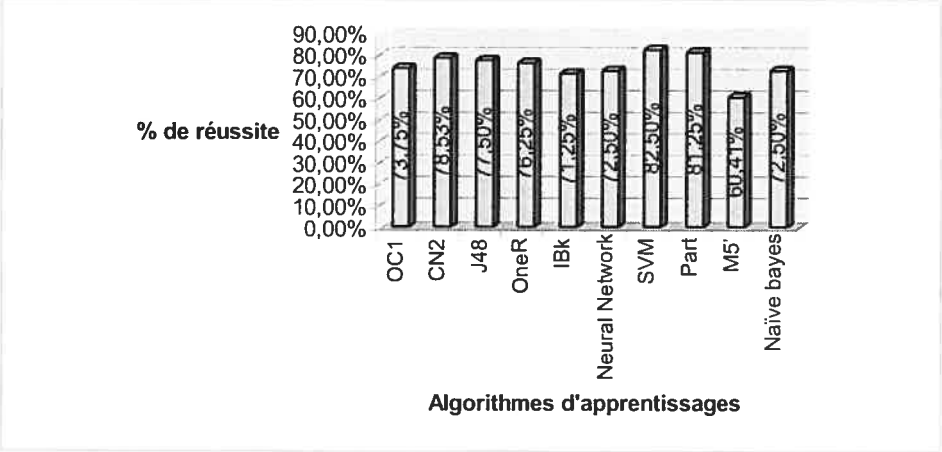


**FIG 5.17: Histogramme hypothèse H3c (cross-validation)**

Concernant la cross-validation, SVM nous offre un taux de succès qui atteint 82.5%. Cela constitue un très bon résultat. Quand à la technique de Split, nous

pouvons remarquer que plusieurs algorithmes ont donné de très bons résultats comme par exemple : J48, OneR, SVM et Part.

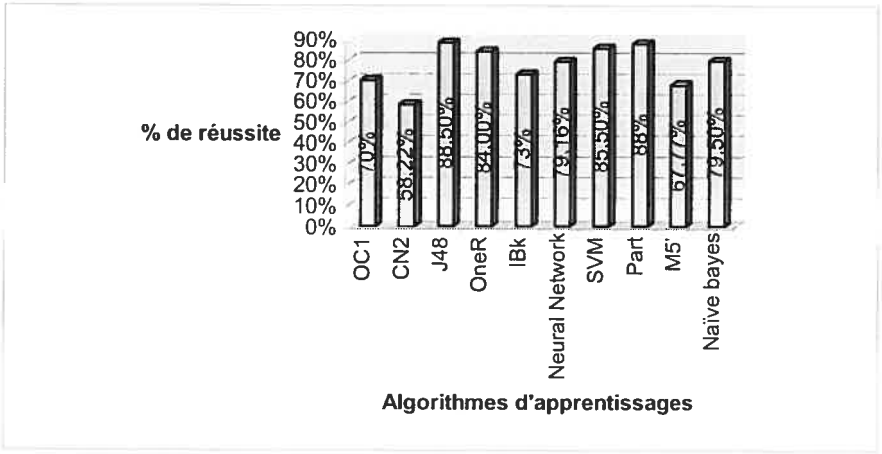


**FIG. 5.18 : Histogramme hypothèse H3c (Split)**

Voici quelques règles générées par CN2 :

| | | |
|---|---|---|
| Règle1 :<br><br>Si 15.50 < **RFC1** < 32.50<br>et ACMIC < 2.50<br>→erreur '1' | Règle 2:<br><br>Si **RFC1** < 125.00<br>et **RFCoo** > 141.50<br>et DAC < 2.50<br>→erreur '1' | Règle 3:<br><br>Si **IH-ICP** > 31.00<br>→erreur '2' |
| Règle 4:<br><br>Si **RFC1** > 5.00<br>Et RFCoo < 16.50<br>Et OCMIC < 2.50<br>→ erreur '2' | | |

**TAB. 5.27: Règles générées par CN2(hypothèse H3c)**

Au niveau des règles de CN2, nous pouvons conclure que les résultats obtenus confirment bien que les attributs déjà sélectionnés par les autres algorithmes sont bons comme par exemple Rfcoo, CBO. Cependant, CN2 met l'accent sur d'autres attributs pour effectuer la classification. Parmi eux, citons : Rfc1 qui apparaît à plusieurs reprises dans les règles et qui apparaît également au niveau de la sélection d'attributs.

-Voici un extrait de l'arbre généré par J48 :

**CBO** <= 1: 2 (4.0)
CBO > 1
| IH-ICP <= 18
| | DCMEC <= 0: 1 (62.0/10.0)
| | DCMEC > 0
| | | ICP <= 9: 2 (4.0)
| | | ICP > 9: 1 (2.0)

J48 se base sur CBO et IH-ICP; ce qui concorde avec les résultats obtenus avec les algorithmes de sélection d'attributs

**-OneR :**

**RFCoo**:
< 16.5 -> b
< 510.0 -> m
>= 510.0 -> b

OneR concorde avec le résultat obtenu par OC1.

### 5.6.4 Comparaison avec d'autres résultats

Nous allons comparer les résultats que nous avons obtenu avec les résultats d'une étude effectuée par Ikonomovski [35] et portant sur les mêmes données mais en utilisant les arbres de décision (C4.5) :

| **Hypothèse** | Hypothèse H3a | | HypothèseH3b | | Hypothèse H3c | |
|---|---|---|---|---|---|---|
| | **CV** | **Split** | **CV** | **Split** | **CV** | **Split** |
| C4.5 | 89.16% | | 84.34% | | 84.34% | |
| CN2 | 87.5% | 89% | | | | |
| Naïve-bayes | | | 77.21% | **87%** | | |
| Part | | | | | 81.25% | **88%** |

**TAB. 5.28: Comparaison de nos résultats avec ceux de C4.5**

Le tableau 5.28 montre que les résultats obtenus avec la cross-validation(cv) sont proches de ceux obtenus avec C4.5. Dans certains cas, les résultats obtenus par la technique du Split (HypothèseH3b, HypothèseH3c) donne de meilleurs résultats par rapport à C4.5.

Eu égard à ces résultats, nous pouvons conclure que certaines mesures sélectionnées pour nos expériences peuvent prédire les classes ayant un potentiel à engendrer des erreurs.

Pour les trois hypothèses énoncées : héritage, cohésion et couplage, on a obtenu des résultats satisfaisants (entre 77% et 87.5%) pour la cross-validation et des résultats variants entre 87 % et 89%. Cela nous permet de confirmer l'existence d'un lien causal entre la liste des mesures sélectionnées lors de chaque expérience et la variable à prédire qui est la propension à engendrer des erreurs.

Nous recommandons les algorithmes CN2 (version améliorée), Part et Naïve Bayes comme les meilleurs algorithmes d'apprentissage pouvant fournir de bonnes prédictions pour les hypothèses de cette expérience.

# Chapitre 6

# Bilan et perspectives

Dans le cadre de cette recherche, nous avons proposé huit hypothèses (conformes au standard IEC 14598) reliant des propriétés de composants orientés-objets avec certaines qualités de logiciels à prédire :

- Une hypothèse relative à la relation entre l'effort de maintenance et des mesures de complexité (Problème 1).
- Quatres hypothèses relatives à la relation entre la réutilisabilité et des mesures d'héritage, de couplage (niveau code et conception) et de complexité (Problème 2) .
- Trois hypothèses relatives à la relation entre la propension à engendrer des erreurs et des mesures d'héritage, de couplage et de cohésion (Problème 3).

Notre travail a consisté en l'évaluation de ces différentes hypothèses en utilisant des algorithmes d'apprentissage.

La première conclusion importante est liée aux taux de succès obtenus avec les algorithmes d'apprentissage utilisés dans nos expériences. Ils sont comparables à ceux déjà obtenu par d'autre techniques et parfois, se sont avérés plus élevés. Par exemple avec CN2 et Neural Network, nous avons pu obtenir des résultats pouvant atteindre respectivement 80% et 85% (en utilisant la cross-validation ) et un taux de 100% (par le Split).

D'un autre coté nous avons prouvé que l'utilisation de la version améliorée de CN2 offre de meilleurs résultats, comme le confirment Clark et Boswell dans leur article [20].

Pour chacune des hypothèses énoncées, nous avons déterminé la liste des attributs pertinents ainsi que les meilleurs modèles de prédiction. Souvent, les résultats ont été conformes à ceux publiés dans d'autres travaux.

Certains algorithmes d'apprentissage comme les réseaux de neurones ne génèrent pas de connaissance (arbre de décision ou règles induites). Il serait donc intéressant de travailler avec différents types d'algorithmes, comme nous l'avons fait au cours de nos expériences. Nous avons remarqué que sous certaines hypothèses, les algorithmes fournissaient des résultats compatibles entre eux même si ces algorithmes sont conçus sur des bases différentes (comme OC1 et J48).

Comme perspective de recherche, il serait intéressant d'étudier d'autres algorithmes d'apprentissage pouvant fournir de meilleurs résultats comme les réseaux bayesiens. Un autre axe envisageable pour la recherche serait de prendre en compte d'autres types de mesures autres que les mesures internes du logiciel. Dans certains cas, la qualité de logiciel pourrait être influencée par des mesures externes comme la compétence des développeurs ou bien la puissance des machines sur lesquelles le logiciel a été développé.

# Bibliographie

[1] Alain R. et Grégory S., Le contrôle de vos projets par l'analyse du code. http://www.cetic.be/article211.html.

[2] Almeida M. A., Lounis H., Melo W. L., An Investigation on the Use of Machine Learned Models for Estimating Software Correction Costs. In 20th IEEE International Conference on Software Engineering, Vol. 18, No. 11, November 1998

[3] Almeida M., Lounis H., Melo W. L., An Investigation on the Use of Machine Learned Models for Estimating Software Correctability. International Journal of Software Engineering and Knowledge Engineering 9(5): 565-594 (1999)

[4] A.Porter and R.Selby. 'Empirically-Guided Software Development Using Metric-Based Classification Trees'.IEEE Software, 7(2): 46-54, March 1990.

[5] Basili V. R., Condon S. E., Emam K. E., Hendrick R. B., Melo W., Characterizing and Modeling the Cost of Rework in a Library of Reusable Software Components.In19th International Conference on Software Engineering, ICSE'97, Boston, Massachusetts, May 1997.

[6] Bensia J., Davis C., Automated Metrics and Object-Oriented Development. Dr. Dobb's journal, december 1997.

[7] Bougrain L., Étude de la construction par réseaux neuromimétiques de représentations interprétables : Application à la prédiction dans le domaine des télécommunications. Phd thesis, Université Nancy 1, 2000.

[8] Boukadoum M., Sahraoui H. A., Lounis H., Machine Learning Approach to Predict Software Evolvability using Fuzzy Binary Trees. In Proc. of International Conference on Artificial Intelligence (IC-AI'2001), 2001.

[9] Briand L., Daly J. Wüst J., A unified Framework for Coupling Measurement in Object-Oriented systems. Technical Report ISERN 96-14, Fraunhofer Institute for Experimental Software Engineering, Germany 1996.

[10] Briand L., Devanbu P., Melo W. L., An Investigation into Coupling Measures for C++, In Proc. of the 19th Intel Conference on Software Engineering, 1997.

[11] Briand L., Devanbu P., Melo W. L., An Investigation into Coupling Measures for C++, In Proc. of the 19th Intel Conference on Software Engineering, 1997.

[12] Briand L., Daly J., Wüst J., A Unified Framework for Coupling Measurement in Object-Oriented Systems. Technical Report ISERN 97-05, Fraunhofer Institute for Experimental Software Engineering, Germany 1997.

[13] Briand L., Daly J., Wüst J., 1998 A Unified Framework for Cohesion Measurement in Object-Oriented Systems. Empirical Software Engineering

Journal, 3 (1), 91-121.

[14] Bieman, J., Kang, B., 1995. Cohesion and Reuse in an Object-Oriented System. Proc. ACM Symp. Software Reusability (SSR'94), 259-262.

[15] Caldiera G., Basili V. R., Identifying and Qualifying Reusable Software Components. IEEE computer, 24(2):61-70, february 1991.

[16] Chapman D., Microsoft Visual C++ 6,Campupress, 2000.

[17] Chidamber S. R., Kemerer C. F., A Metric Suite for Object Oriented Design. IEEE Transactions on Software Engineering, 20 (6) : 476-493, june 1994.

[18] Chidamber , S.R, and C.F. Kemerer. Towards a Metric Suite for Object-Oriented Design, Proceedings : OOPSLA '91, Phoenix, AZ, July 1991, pp. 197-211.

[19] Churcher N. I., Shepperd M. J., Comments on A metrics Suite Object-Oriented Design. IEEE Transactions on Software Engineering, Vol. 21, No. 3, March 1995.

[20] Clark P.,Boswell R.,Rule Induction With CN2: Some Recent Improvements,1991.

[21] Clark P.,Niblett T., The CN2 Induction Algorithm, Machine Learning; 3:261- 83,1989. http://citeseer.nj.nec.com/clark89cn.html

[22] Cornuejols A., Une nouvelle méthode d'apprentissage: Les S.V.M. Séparateur à Vaste marge, Université de Paris Sud, Juin 2002.

[23] Coulange B., Réutilisabilité du logiciel, MIPS, Masson, 1996.

[24] Devanbu P., Eaves L. E., How to write a GEN++ spécification. AT&T, june 1994.

[25] D.Doubleday 'ASAP: An Ada Static Source Code Analyzer Program'. CS-TR-1895, Computer Science Dept, University of Maryland, College Park, MD, 1995

[26] Etzkorn, L. and Davis, C. G. "Automatically Identifying Reusable OO Legacy Code." IEEE Computer, 30(10): 66-71, 1997.

[27] G. Booch. Object-Oriented Design With Applications. Benjamin/Cummings Publishing Company Inc., Santa Clara, California, 1994

[28] Hakim Lounis, Houari A. Sahraoui, Walcélio L. Melo: Vers un modèle de prédiction de la qualité du logiciel pour les systèmes à objets. L'objet - logiciel, bases de données, réseaux, volume 4, numéro 4, décembre 1998.

[29] Haykin, S.(1994), Neural Networks : A Comprehensive Foundation, NY , Macmillan, p.2

[30] Henderson-Sellers B., Some Metrics for Object Oriented Software Engineering, in Proc. of the International Conference on Technology of Object-Oriented Languages and Systems, TOOLS 6 Pacific 1991, pp. 131-139, TOOLS USA, 1991.

[31] Henderson-Sellers B., Software Metrics, Prentice Hall, Hemel Hempstaed, U.K, 1996.

[32] Hitz, M., and B. Montazeri. Chidamber and Kemerer's Metric Suite : A Measurement Theory Perspective, IEEE Transactions on Software Engineering. Vol. 4, April 1996, pp. 267- 271.

[33] Hitz, M., and B. Montazeri. Measuring Coupling and Cohesion in Object-Oriented Systems.Proc.Int. Symposium on Applied Corporate Computing, Monterry, Mexico, 1995.
http://www.pri.univie.ac.at/~hitz/papers/ESEC95.ps

[34] Halstead, Maurice H. Elements of Software Science, Operating, and Programming Systems Series Volume 7. New York, NY: Elsevier, 1977.

[35] Ikonomovski S. V., Detection of Faulty Components in Object-Oriented Systems using Design Metrics and Machine Learning Algorithm, Mc Gill University, Montreal, 1998.

[36] J. Bansiya, and C. Davis. Using Automated Metrics to Track Object-Oriented Development.

[37] J. R. Quinlan. Discovering rules by induction from large numbers of examples: a case study. In D. Michie, editor, *Expert systems in the micro-electronic age*. Edinburg University Press, 1979.

[38] J. R. Quinlan. Induction of deciscion trees. In *Machine learning*, volume 1, pages 81-106. Kluwer Academic Publishers, 1986.

[39] Kirby et al., Engineering in History, 1990.

[40] Lee Y. S., Liang B. S., Wu S. F., Wang F. J., Measuring the Coupling and Cohesion of an Cbject-Criented Program Based on Information Flow. In

Proceedings of International Conference on Software Quality, Maribor, Slovenia, 1995.

[41] Li, W., and S. Henry. Maintenance Metrics for the Object-Oriented Paradigm. Proceedingsof the First International Software Metrics Symposium, Baltimore, MD, May 1993, pp. 52-60.

[42] Li W., Henry S., Object Oriented Metrics that Predicts Maintainability. System and Software, Vol. 23, N 2, 1993.

[43] Li, W., S. Henry, D. Kafura, and R. Schulman. Measuring Object-oriented Design. Journal of Object-Oriented Programming, Vol. 8, No. 4, July 1995, pp. 48-55.

[44] L. Hakim., L. Ait-Mehedine., Machine-learning Techniques for Software Product Quality Assessment, 2004

[45] Lounis H., W. Melo L., Sahraoui H., Identifying and Measuring Coupling in Modular Systems. Centre de Recherche Informatique de Montréal, Montréal, Canada 1997.

[46] Lorenz, M., Kidd, J., Object-Oriented Software Metrics. Prentice Hall Object-Oriented Series, Englewood Cliffs, N.J., 1994.

[47] Mao Y., A Metric Based Detection of Reusable Object-Oriented Software Components Using Machine Learning Algorithm. Master thesis, Mc Gill University, Montreal, 1998.

[48] Mac Culloch, W.S. Pitts, "A Logical Calculus of the Ideas Immanent In nervous Activity" Bull. Math. Biophys. 5, 115-133 – 1943

[49] M.A. Ochs.M System:Calculating Software Metrics from C++ Source Code. IESE Fraunhofer Report No, 005.98/E February 1998.

[50] McCall J. A., Richards P. K., Walters G. F., Factors in Software Quality, General Electric Company, RADC-TR77-369, Three volumes, Rome Air Development Center, November 1977.

[51] Mitchell T.M.,'Machine Learning',McGraw-Hill Publishing Compagny, McGraw-Hill Series in Computer Science(Artificial Intelligence),1997.
http://www-2.cs.cmu.edu/~tom/mlbook-chapter-slides.html

[52] Micro Application , Visual C++ 6.0, 2000.

[53] Myers, J.G.: The Art of Software Testing. John Wiley & Sons, Inc. (1979)

[54] Myrthy S. K.,Kassif S., Salzberg S., A System for Induction of Oblique Decision Trees, Journal of Artificial Intelligence Research 2, 1-33, August 1994. http://www.cs.jhu.edu/~salzberg/announce-oc1.html

[55] Oman P., Pfleeger S. L., Applying Software Metrics, 1994.

[56] Price M. W., Steven S., Demurjian A., Analyzing and Measuring Reusability in Object-Oriented Design. In OOPSLA, October 1997.

[57] Price M. W., Steven S., Demurjian A., Analyzing and Measuring Reusability in Object-Oriented Design. In OOPSLA, October 1997.

[58] Quinlan J. R.. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, Sao Mateo, CA, 1993.

[59] R. S. Michalski and J. Larson. Incremental generation of VL1 hypotheses: The Underlying Methodology and the Description of Program AQ11. ISG 83-5, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, Urbana, 1983.

[60] S.Chidamber, and C.kemerer. A Metric Suite for Objetc Oriented Design.IEEE Transactions on Software Engineering, Vol.20, No. 6, June 1994

[61] Selby R. W., Basili V. R., Analyzing Error-Prone System Structure. IEEE Transactions on Software Engineering, Vol. 17, No. 2, February 1991.

[62] Siveton, V., Extraction de Connaissances hydriques au moyen de diverses méthodes supervisées, Montréal, Juin 2002.

[63] Tegarden, D., Sheetz, S., Monarchi, D., A Software Complexity Model of Object-Oriented Systems. Decision Support Systems, 13 (3-4), 241-262.

[64] V.Basili, L.Briand, S. Condon, Y. M. Kim, W.L Melo, and J.D Valett. Understanding and Predicting the process of Software Maintenance Releases. In 8th IEEE International Conference of Software Engineering, Berlin, Germany, 1996.

[65] Vladimir N. Vapnik, The Nature of Statistical Learning Theory. Springer, 1995.

[66] Witten I. H., Frank E., Data Mining , Practical Machine learning Tools and Techniques With Java Implementations.1999

[67] Weiss, Kulikowski, Computer Systems that Learn (context), 1991.

[68] W. Just, L.C.Briand, H.Lounis, Replicated Case Studies for Investigating Quality Factors in Object-Oriented Designs, In Empirical Soft. Engineering, an Int. Journal, 6 (1):11-58, 2001, Kluwer Academie Publishers.

# Annexe A

# Les arbres de décision

Les arbres de décision sont composés d'une structure hiérarchique. Les nœuds internes de l'arbre sont des tests sur les attributs, les feuilles sont les classes. Lorsque les tests sont binaires, le fils gauche correspond à une réponse positive au test et le fils droit à une réponse négative. Cette structure est construite grâce à des méthodes d'apprentissage par induction à partir d'exemples.

Un arbre de décision se construit à partir d'un ensemble d'apprentissage par raffinement de sa structure. Un ensemble de questions sur les attributs est construit afin de partitionner l'ensemble d'apprentissage en sous-ensembles qui deviennent de plus en plus petits jusqu'à ne contenir à la fin que des observations relatives à une seule classe. Les résultats des tests forment les branches de l'arbre et chaque sous-ensemble forme les feuilles. Le classement d'un nouvel exemple se fait en parcourant un chemin qui part de la racine pour aboutir à une feuille : l'exemple appartient à la classe qui correspond aux exemples de la feuille. Ainsi, nous pouvons résumer qu'un arbre de décision est une représentation graphique d'une procédure de classification.

Nous présentons dans se qui suit une problématique simple aboutissant à la construction d'un arbre de décision(Voir FIG.A.1). Cette problématique est issue des travaux de Quinlan [58].

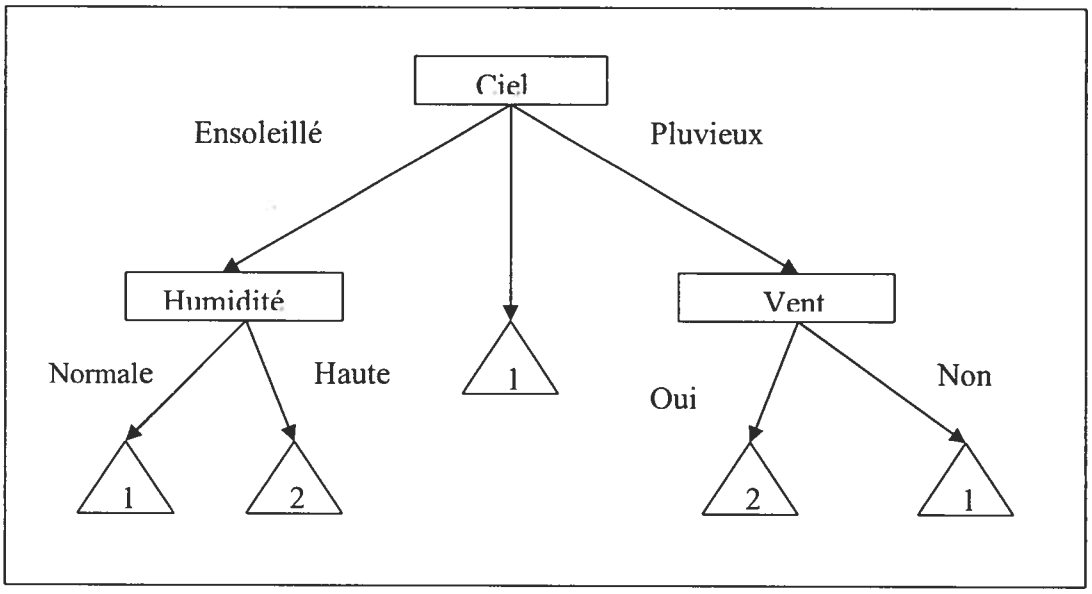


**FIG. A.1 Exemple d'arbre de décision [50]**

On peut prendre comme exemple de règles :

Si ((Ciel = Ensoleillé et humidité=Normale)
Ou (Ciel = Couvert)
Ou (Ciel= Pluvieux et Vent=Faux))
Alors classe =1

L'apprentissage est fait à partir d'une base d'exemples qui possèdent un certain nombre d'attributs significatifs, par exemple la température, le vent, l'humidité, etc. Chaque exemple associe des valeurs particulières à chaque attribut, et

comme cette méthode est une méthode d'apprentissage supervisée, chaque exemple est associe à une classe particulière.

Le principe de construction des arbres est le suivant : Il suffit de choisir un attribut parmi les attributs non sélectionnés, suivant le résultat d'un test de mérite, et de créer un nœud portant un test sur cet attribut. Pour chaque classe d'équivalence ainsi induite, il s'agit d'opérer le traitement suivant :Si tous les exemples de cette classe d'équivalence appartiennent à la même classe (les classes météorologique décrites dans le tableau ci-dessus), alors il s'agit de créer une feuille correspondant; si tous les exemples de la classe d'équivalence considérée ne sont pas dans la même classe, alors il faut réitérer ce processus en enlevant l'attribut précédemment considère des attributs à sélectionner.

Nous pouvons citer comme exemple d'algorithme générant des arbres de décision, ID3 [37],[38] et C4.5 [58].

C4.5 est une référence de base dans les études sur les arbres de décision. Il présente des propositions pour traiter et améliorer la discrétisations des variables quantitatives, la prise en compte des coûts associés au choix de chaque attribut, la prise en compte par la fonction de sélection d'attributs, l'élagage de l'arbre à la fin du processus d'apprentissage, à partir d'un ensemble d'exemples de test de généralisation et l'explicitation de règles symboliques du type Si-Alors à partir des arbres de décision.

# Annexe B

# CN2 et les taux obtenus

Le tableau B.1 démontre les taux de succès obtenus par Peter Clark et Robin Boswel [20] à travers leurs expériences qu'ils ont effectué afin de prouver l'efficacité de la nouvelle version de Cn2.

Notons que chaque ligne du tableau correspond à un domaine d'expérimentation différent.

| Domaine ? | | | Cn2(Laplace) | | | | |
|---|---|---|---|---|---|---|---|
| | Cn2(Entropie) | | Règle ordonnée | | Règle non rdonnée | | C4.5 |
| Seuil de signification? | 0% | 99.5% | 0% | 99.5% | 0% | 99.5% | |
| Lymphography | 71.5±6.3 | 68.4±8.6 | 79.6±5.7 | 74.4±5.9 | 81.7±4.3 | 6.5±5.3 | 76.4±6.2 |
| Poteau et chariot | 52.5±1.9 | 52.2±1.7 | 70.6±3.1 | 67.9±3.3 | 72.0±2.9 | 63.0±3.2 | 74.3±2.0 |
| Haricot de soja | 74.7±6.7 | 54.2±6.5 | 82.7±3.9 | 57.5±4.6 | 81.6±3.8 | 76.1±4.4 | 80.0±3.6 |
| maladie de coeurC | 66.3±8.5 | 67.1±9.2 | 75.4±3.6 | 76.1±4.4 | 76.7±3.9 | 76.6±3.7 | 76.4±4.5 |
| maladie de coeurH | 73.0±4.6 | 81.6±3.4 | 75.0±3.8 | 74.9±4.7 | 78.8±4.1 | 77.8±3.9 | 78.0±5.5 |
| Verre | 45.2±8.1 | 44.4±7.7 | 58.5±5.0 | 56.9±7.7 | 65.5±5.6 | 61.6±8.3 | 64.2±5.1 |
| tumeur primaire | 35.6±5.2 | 33.0±3.5 | 49.7±9.8 | 38.7±5.3 | 45.8±3.6 | 41.4±5.8 | 39.0±4.0 |
| Enregistrement de vote | 93.6±1.8 | 94.0±1.8 | 94.8±1.7 | 92.8±1.8 | 94.8±1.8 | 93.3±2.1 | 95.6±1.1 |
| Thyroïde | 95.6±0.7 | 95.6±0.9 | 96.3±0.7 | 96.3±0.5 | 96.6±0.9 | 96.1±1.2 | 96.4±0.9 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| cancer de sein | 69.0±3.6 | 68.7±4.3 | 65.1±5.3 | 64.2±7.6 | 73.0±4.5 | 70.8±3.5 | 72.1±3.7 |
| Hépatite | 71.3±5.2 | 77.5±5.6 | 77.6±5.9 | 78.1±5.9 | 80.1±5.7 | 80.8±4.5 | 79.3±5.8 |
| Echocardio | 63.9±5.4 | 67.5±5.6 | 62.3±5.1 | 63.2±7.7 | 66.6±7.3 | 69.4±6.8 | 63.6±5.3 |
| Moyenne | 67.7 | 67.0 | 74.0 | 70.1 | 76.1 | 73.6 | 74.6 |

**TAB. B.1 : Taux de succès obtenu par CN2.**