

UNIVERSITE SIDI MOHAMED BEN ABDELLAH
FACULTÉ DES SCIENCES ET TECHNIQUES FÈS
DÉPARTEMENT D'INFORMATIQUE



PROJET DE FIN D'ÉTUDES

**MASTER SCIENCES ET TECHNIQUES
SYSTÈMES INTELLIGENTS & RÉSEAUX**

**LES MESURES LOGICIELS ET LA PRÉDICTION DES DÉFAUTS
POUR LES APPLICATIONS ORIENTÉS OBJETS**



LIEU DE STAGE : LABORATOIRE DES SYSTÈMES INTELLIGENTS ET APPLICATIONS

RÉALISÉ PAR : EL MAZOURI FATIMA ZAHRA
EL HOUARI MERIAME

SOUTENU LE : 25 /06 /2015

ENCADRÉ PAR :

MME L.LAMRINI

DEVANT LE JURY COMPOSÉ DE :

PR. L.LAMRINI
PR. K. ZENKOUAR
PR. A. BENABBOU
PR. A.M.CHAOUKI

ANNÉE UNIVERSITAIRE 2014-2015

Remerciement

Avant toute chose, on remercie Allah, le Tout-Puissant, qui nous a donné la force et le courage pour effectuer ce travail.

Nous tenons à remercier dans un premier temps, toute l'équipe pédagogique de la faculté des sciences et techniques de Fès et les intervenants professionnels responsables de la formation Système Intelligent et Réseau.

Avant d'entamer ce rapport, nous profitons de l'occasion pour remercier tout d'abord notre professeur Madame **LAMRINI LOUBNA** qui n'a pas cessé de nous encourager pendant la durée du projet, ainsi pour sa générosité en matière de formation et d'encadrement. Nous la remercions également pour l'aide et les conseils concernant les missions évoquées dans ce rapport, qu'elle nous a apporté lors des différents suivis, et la confiance qu'elle nous a témoignée.

Nous tenons à remercier nos professeurs qui nous ont incitées à travailler en mettant à notre disposition leurs expériences et leurs compétences.

Nous adressons nos remerciements particuliers aux membres du jury de soutenance président Mr K. Zenkouar, Mr A. BENABBOU et Mr A.M.CHAOUKI qui ont aimablement accepté d'évaluer ce travail.

Nous adressons un grand remerciement à nos mères qui n'ont pas cessé à nous préparer les bonnes conditions pour continuer nos études universitaires, et à nos chers pères pour tout le soutien qu'il nous a apporté grâce à ses conseils et ses encouragements.

Enfin, nous tenons à remercier toutes les personnes qui nous ont conseillées et relues lors de la rédaction de ce rapport de stage : nos familles, nos amies et nos camarades de promotion.

Nous espérons que vous prendrez autant de plaisir à lire ce rapport que nous en avons pris durant tout le déroulement de ce projet.

Résumé

La construction de modèles prédictifs d'un logiciel bien avant sa mise en œuvre est maintenant une pratique largement menée dans l'industrie du logiciel. Il est bien connu que la production d'un système de qualité nécessite un grand soin dans les premières phases du développement. Nous nous intéressons à la phase de conception qui se caractérise par la construction d'un élément crucial pour la description du système, qui est le diagramme de classes, donc la qualité du diagramme de classe a un impact important sur la qualité du logiciel qui sera implémenté. L'objectifs de cette étude est de sélectionner un ensemble de métriques extraites à partir d'un diagramme de classe pour des applications orientées objets, afin d'appliquer des modèles de prédiction de défauts lors de l'étape initiale d'un développement orienté-objet.

MOTS-CLES: Métrique orienté-objet, qualité du logiciel orienté-objet, diagramme de classe, modèles de prédiction de défauts.

Abstract

Building software models before implementing them has become widely accepted in the software industry. Object models, graphically represented by class diagrams, lay the foundation for all later design work. So, their quality can have a significant impact on the quality of the software which is ultimately implemented. It is widely recognised that the production of better software requires the improvement of early development phases and the artifacts they produce. In this paper, we will introduce and analyse a set of an existent object oriented metrics that can be applied for assessing class diagrams complexity at the initial phases of the object oriented development life cycle.

KEY WORDS: object oriented metrics, object oriented software quality, class diagram, bugs prediction model.

Table des matières

Chapitre1 : Qualité logiciel	13
I. Introduction.....	14
II. Génie logiciel	14
1. Définition	14
2. Objectif	14
3. Principe.....	14
4. Cycle de vie d'un logiciel.....	14
5. La crise du logiciel.....	15
III. Qualité du logiciel.....	15
1. Définition	15
2. Les principaux facteurs et comment pouvoir l'améliorer.	15
3. Amélioration.....	16
3.1. Relation entre métriques et qualité	16
IV. Conclusion	16
Chapitre2 : Les métriques logicielles.....	17
I. Introduction.....	18
II. Les métriques	18
1. Définitions	18
2. Historique	18
3. Intérêt.....	20
4. Les métriques	21
4.1. Les métriques logicielles.....	22
5.2. Les métriques orientées objets	25
III. Conclusion	28
Chapitre3 : Métriques du diagramme de classes.....	29
I. Introduction.....	30
I. Le langage de modélisation unifié.....	30
1. Définition	30
2. L'intérêt	30
II. Le diagramme de classes.....	30
1. Définition	30
2. L'intérêt	30
3. Les composants du diagramme de classes.....	31

III.	Les métriques du diagramme de classe	33
1.	Les métriques de Chidamber et Kermer.....	34
2.	Les métriques MOOD (Metrics for Object Oriented Design)	34
3.	Les métriques de Li et Henry	34
4.	Métrique de Lorenz et Kidd	34
5.	Les métriques de Briand	36
6.	Métriques de Harrison.....	36
7.	Métriques de Genero	36
8.	Métriques de Bansiya	38
IV.	Conclusion	39
	Chapitre4 : Aide à la décision	40
I.	Introduction.....	41
II.	L'aide à la décision.....	41
1.	Définition	41
2.	L'historique.....	41
3.	Domaines d'application.....	42
4.	Rôle d'aide à la décision	42
5.	Intérêt d'outil d'aide à la décision.....	42
4.1.	Des outils d'aide à la discision	43
4.2.	Les avantages.....	43
4.3.	Les inconvénients	43
6.	L'Informatique dans l'aide à la décision.....	44
III.	Datamining	45
1.	Définition	45
2.	Les domaines d'application	45
3.	Les processus de Data Mining	46
4.	Les tâches de Data Mining.....	46
4.1.	Classification	46
4.2.	Estimation.....	47
4.3.	Prédiction	47
4.4.	Groupement par similitudes.....	47
4.5.	Segmentation (ou clusterisation)	48
4.6.	Description.....	48
4.7.	Optimisation	48

IV.	Les techniques les plus appropriées au Data Mining	48
1.	Réseaux de neurones	48
1.1.	Définition	48
1.2.	Histoire	49
1.3.	Application.....	49
1.4.	Modèle biologique.....	49
1.5.	Neurone formel	50
1.6.	Le perceptron	50
1.7.	Le Perceptron multicouche	54
2.	Arbre de décision.....	56
2.1.	Définition	56
2.2.	Constitution	56
2.3.	But d'arbre de décision	57
2.4.	Les domaines d'application	58
2.5.	La construction d'un arbre de décision	58
2.6.	Les avantages.....	64
2.7.	Les inconvénients	64
2.8.	Exemple des méthodes qui aide à la construction d'un arbre	65
3.	SVM.....	65
3.1.	Définition	65
3.2.	Principe de fonctionnement général.....	65
3.3.	L'historique	66
3.4.	Notions de base	66
3.5.	Les domaines d'applications.....	68
3.6.	Les avantages	68
3.7.	Inconvénients	69
4.	Les Réseaux bayésiens	69
4.1	Définition	69
4.2.	Classification naïve bayésienne	69
4.3.	Exemples.....	69
V.	Conclusion	71
	Chapitre5 : Réalisation	72
I.	Introduction.....	73
II.	Base de données.....	73

1. Quelques bases de données du génie logiciel	73
2. Base de données utilisée	77
2.1. Définition	77
2.2. Les raisons de choix de PROMISE	77
2.3. Les métriques de PROMISE	78
2.4. Les métriques utilisées par notre application	80
2.5. Vision sur le site de PROMISE	81
III. Outils de développement	85
1. WEKA	85
1.1 Définition	85
1.2. But ultime	85
1.3. Version utilisée	85
1.4. Composants de Weka	86
2. NetBeans	92
2.1. Définition	92
2.2. But ultime	92
2.3. Version utilisée	92
3. Vue global sur l'application	92
3.1. Objectif	92
3.2. Description	93
4. Capture d'écran	94
5. Étude de convergence des méthodes utilisées	95
IV. Conclusion	96
Conclusion et perspectives	97
Références	98

Liste des figures

Figure 1:Graphe de mesure de complexité d'une classe.....	24
Figure 2:Présentation graphique d'un paquetage d'un diagramme de classe	31
Figure 3:La structure générale d'une classe	31
Figure 4:Exemple des niveaux de visibilité	32
Figure 5:Exemple d'héritage	32
Figure 6:Exemple d'une relation d'association.....	32
Figure 7:Exemple d'une relation d'agrégation	33
Figure 8:Exemple d'une relation de composition.....	33
Figure 9: Exemple d'une relation de dépendance	33
Figure 10:Représentation d'un neurone formel.....	48
Figure 11:Neurone biologique	49
Figure 13:Représentation des différentes fonctions de transfert.....	50
Figure 12:Neurone formel	50
Figure 14:Exemples du perceptron.....	51
Figure 15:Exemple d'apprentissage.....	53
Figure 16:Exemple de résultat du calcul.....	53
Figure 17:Présentation du poids de connexion	53
Figure 18:Exemple du perceptron multi couche	54
Figure 19:Modèle de neurone j	55
Figure 20:Réseau de neurone et reconnaissance de forme	56
Figure 21:Représentation d'un arbre de décision	57
Figure 22:Représente le but d'arbre de décision	57
Figure 23: Le comportement de joueur de Tennis à partir de prévisions météorologiques	58
Figure 24:L'attribut humidité.....	60
Figure 25:L'attribut vent	60
Figure 26:L'attribut temps	60
Figure 27:L'attribut température.....	61
Figure 28:La valeur du gain pour chaque attribut	61
Figure 29:Construction des sommets	62
Figure 30:Construction du premier sommet	62
Figure 31:Construction du deuxième sommet	63
Figure 32:L'arbre obtenu	63
Figure 33:L'arbre obtenu par logiciel weka	64
Figure 34:L'hyperplan H qui sépare les deux ensembles de points.	67
Figure 35:Les vecteurs de support.....	67
Figure 36:Hyperplan optimal, vecteurs de support et marge maximale.....	68
Figure 37:Vision sur le site de PROMISE	82

Figure 39:Les informations disponibles pour chaque base de données	83
Figure 38:Les bases de données trouvées	83
Figure 40:La fenêtre de téléchargement de la base de données zuzel	84
Figure 41:Exemple d'une base de données PROMISE (zuzel)	84
Figure 42:Environnement Weka	85
Figure 43:Capture d'écran du module Explorer de Weka	86
Figure 44:Le site de conversation CSV à ARFF	88
Figure 45:Chargement de l'ensemble des données	89
Figure 46:Familles d'algorithmes d'apprentissages automatiques Supervisés	90
Figure 47:Résultats de l'algorithme J48.....	91
Figure 48:Version de NetBeans utilisé	92
Figure 49:Fenêtre principale de l'application	94
Figure 50:Choix de méthode d'apprentissage	94
Figure 51:Fenêtre d'une étape de l'application.....	95
Figure 52:Exemple de résultat avec les arbres de décision	95

Liste des tableaux

Tableau 1:Tableau résume les différentes métriques de Briand	36
Tableau 2:La validation théorique des métriques utilisant Briand et al.....	38
Tableau 3:Les critères de la base de données	58
Tableau 4:Résultat obtenu à l'aide de l'algorithme naïve bayes.....	70
Tableau 5:Résultat de calcul	71
Tableau 6:Description des bases de données du génie logiciel	73
Tableau 7:Les métriques utilisées.....	80
Tableau 8:Résultat d'étude de convergence de chaque méthode.....	96

Introduction générale

Les grands projets de logiciels sont sujets à des risques de qualité, ayant des modules défectueux qui vont entraîner des échecs lors de l'exécution du logiciel. La plupart de ces bugs proviennent d'un manque d'informations dans une phase précoce de développement de logiciels ce qui influence sur l'effort de développement, l'allocation des ressources de développement, la planification des activités de développement ainsi que **des défauts qui peuvent apparaître lors du processus de développement.**

La qualité des systèmes de logiciels orientés objet dépend fortement de la précision de la spécification des prescriptions. Par conséquent, un effort important devrait se concentrer sur l'amélioration des modèles produites dans les premières phases de cycle de vie de développement.

Nous étudions une approche de précision de l'estimation de taux de défaut au début du cycle de vie de développement de logiciel orienté objet, en utilisant des métriques logicielles recueillies à partir de diagramme de classes UML qui est un élément crucial dans la phase de conception d'un logiciel.

Pour pouvoir faire cette étude, nous avons utilisé la fameuse base de données logicielle PROMISE (PRedictor Models In Software Engineering), qui est un référentiel international dans le génie logiciel. Cette base de données offre une collection de données accessibles au public à servir les chercheurs dans la construction de modèles prédictifs de logiciels et la communauté de génie logiciel au sens large.

Afin de bien appliquer des modèles de prédiction nous avons utilisé quatre méthodes du DataMining comme une technique d'aide à la décision, à savoir les arbres de décision, les réseaux de neurones, SVM et les réseaux bayesiens. Elles sont choisies en raison de leurs capacités de classification.

Notre rapport s'articule autour de cinq principaux chapitres, le premier chapitre est consacré à présenter une vision générale sur la qualité d'un logiciel, suivi d'une présentation des différentes métriques logicielles dans le chapitre 2. Dans le chapitre 3 nous présentons une description des métriques sélectionnées à partir du diagramme de classe. Le chapitre 4 est réservé à la présentation des techniques d'aide à la décision, le dernier chapitre est consacré à la partie réalisation, et nous avons terminé notre rapport par une conclusion et des perspectives.

Chapitre1 : Qualité logiciel

I. Introduction

L'information est aujourd'hui une ressource stratégique pour la plupart des entreprises, dans lesquelles de très nombreuses activités reposent sur l'exploitation d'applications informatiques. Pour ces entreprises, la fiabilité de leur système informatique et la qualité des logiciels utilisés sont donc cruciales.

II. Génie logiciel

1. Définition

Ensemble des méthodes, des techniques et des outils dédiés à la conception, au développement et à la maintenance des systèmes informatiques [1].

2. Objectif

Avoir des procédures systématiques pour des logiciels de grande taille afin que

- la spécification corresponde aux besoins réels du client,
- le logiciel respecte sa spécification,
- Les délais et les coûts alloués à la réalisation soient respectés [1].

3. Principe

Les principes utilisés dans le Génie Logiciel sont :

- **Généralisation** : regroupement d'un ensemble de fonctionnalités semblables en une fonctionnalité paramétrable (généricité, héritage).
- **Structuration** : façon de décomposer un logiciel.
- **Abstraction** : mécanisme qui permet de présenter un contexte en exprimant les éléments pertinents et en omettant ceux qui ne le sont pas.
- **Modularité** : décomposition d'un logiciel en composants discrets.
- **Documentation** : gestion des documents incluant leur identification, acquisition, production, stockage et distribution.
- **Vérification** : détermination du respect des spécifications établies sur la base des besoins identifiés.

4. Cycle de vie d'un logiciel

Le « cycle de vie d'un logiciel » (en anglais software lifecycle), désigne toutes les étapes du développement d'un logiciel, de sa conception à sa disparition.

Le cycle de vie du logiciel comprend généralement les activités suivantes :

- **Définition des objectifs**, consistant à définir la finalité du projet et son inscription dans une stratégie globale.
- **Analyse des besoins et faisabilité**, c'est-à-dire l'expression, le recueil et la formalisation des besoins du demandeur (le client) et de l'ensemble des contraintes.
- **Conception générale**, Il s'agit de l'élaboration des spécifications de l'architecture générale du logiciel.

- **Conception détaillée**, consistant à définir précisément chaque sous-ensemble du logiciel.
- **Codage (Implémentation ou programmation)**, soit la traduction dans un langage de programmation des fonctionnalités définies lors de phases de conception.
- **Tests unitaires**, permettant de vérifier individuellement que chaque sous-ensemble du logiciel est implémenté conformément aux spécifications.
- **Intégration**, dont l'objectif est de s'assurer de l'interfaçage des différents éléments (modules) du logiciel. Elle fait l'objet de tests d'intégration consignés dans un document.
- **Qualification**, c'est-à-dire la vérification de la conformité du logiciel aux spécifications initiales.
- **Documentation**, visant à produire les informations nécessaires pour l'utilisation du logiciel et pour des développements ultérieurs.
- **Mise en production**,
- **Maintenance**, comprenant toutes les actions correctives (maintenance corrective) et évolutives (maintenance évolutive) sur le logiciel.

La séquence et la présence de chacune de ces activités dans le cycle de vie dépend du choix d'un modèle de cycle de vie entre le client et l'équipe de développement. Parmi les modèles les plus fréquents, nous trouvons : modèle en cascade, modèle en V, modèle en spirale [2].

5. La crise du logiciel

Un phénomène de baisse des prix du matériel informatique et d'augmentation des prix du logiciel, accompagné d'une baisse de la qualité des logiciels a été identifiés à la fin des années 1960 et nommé la « crise du logiciel ». Les symptômes de cette crise sont les suivants :

- Délais de livraison des logiciels non respectés,
- Budget non respecté,
- Le logiciel ne répond pas aux besoins de l'utilisateur ou du client,
- Le logiciel est difficile à l'utiliser, le maintenir, et de le faire évoluer [1].

III. Qualité du logiciel

1. Définition

La qualité du logiciel est définie par son aptitude à satisfaire les besoins des utilisateurs. Elle est définie par l'ANSI (American National Standards Institute) comme "l'ensemble des attributs et caractéristiques d'un produit ou d'un service qui portent sur sa capacité à satisfaire des besoins donnés".

2. Les principaux facteurs et comment pouvoir l'améliorer.

Les principaux facteurs de qualité d'un logiciel sont la conformité aux besoins, la fiabilité, l'ergonomie (dont la facilité d'emploi), la flexibilité, la maintenabilité, l'intégrité et la disponibilité.

Au vu de son utilisateur, un logiciel de qualité doit donc présenter ces caractéristiques sans que son efficacité, ses performances (temps de réponse, place mémoire minimum, ...) en pâtissent.

Une définition des principaux facteurs de qualité d'un logiciel est proposée ci-dessous :

- **Disponibilité** : Aptitude du logiciel à assurer sa fonction pendant une période de temps donnée.
- **Ergonomie** : l'ergonomie vise l'adaptation des machines et du travail à l'homme en permettant la conception d'outils qui puissent être utilisés avec le maximum de confort, d'efficacité et de sécurité. On considère que la convivialité, la facilité d'apprentissage pour l'utilisateur font partie de l'ergonomie.
- **Fiabilité** : Aptitude du logiciel à accomplir sans défaillance l'ensemble des fonctions spécifiées, à fonctionner dans des conditions anormales sans mettre en cause ni les informations du système, ni leur cohérence.
- **Flexibilité** : Caractère d'un logiciel qui définit la facilité avec laquelle des fonctions peuvent être ajoutées, supprimées ou modifiées dans un programme opérationnel.
- **Intégrité** : Faculté d'un logiciel à être protégé contre des altérations ou contre l'accès par des utilisateurs non autorisés.
- **Maintenabilité** : Caractère d'un logiciel qui définit la facilité avec laquelle un défaut peut être localisé, identifié et corrigé [3].

3. Amélioration

3.1. Relation entre métriques et qualité

Beaucoup d'études sont menées pour trouver des méthodes efficaces pour l'amélioration de la qualité : plusieurs métriques, plusieurs modèles de qualité et plusieurs normes ont été proposés afin de contribuer à la production de logiciel qualité.

La mesure est utilisée souvent pour une amélioration systématique du fait que :

- La mesure décrit quantitativement l'état courant.
- La connaissance de l'état courant permet de définir des objectifs quantitatifs réalistes d'amélioration.
- La connaissance de l'état actuel permet d'identifier les points forts et les points faibles du processus utilisé.
- La connaissance des points faibles du processus permet d'identifier les changements à faire pour l'améliorer.
- L'impact d'un changement ne peut être mesuré que s'il existe une base quantitative permettant la comparaison.

IV. Conclusion

C'est ce que nous avons adopté comme idée dans notre recherche. Dans les chapitres suivants nous allons voir les métriques utilisées pour la prédiction des défauts logiciels orientée objet dans une phase précoce de développement, afin d'améliorer sa qualité [4].

Chapitre2 : Les métriques logicielles

I. Introduction

Ce chapitre présente les métriques orientées objets selon des catégories bien spécifiques. Ces métriques ont été proposées pour mesurer les propriétés des systèmes orientés objet comme la taille, le couplage et la cohésion.

II. Les métriques

1. Définitions

En langage naturel, une métrique est un mot polysémique, comme théorie de la mesure dans un espace dans le Petit Robert.

En informatique, une métrique logicielle est une compilation de mesures issues des propriétés techniques ou fonctionnelles appliquée à la production logicielle, elle est un indicateur d'avancement ou de qualité de développement logiciels.

Selon IEEE 1061 de 1992

Selon IEEE (Institute of Electrical and Electronics Engineers) 1061 de 1992, une métrique est une fonction dont les entrées sont les données du logiciel et dont la sortie est une valeur numérique qui peut être interprétée comme le degré auquel un élément tend vers un facteur de qualité.

Selon ISO/IEC 9126 [2003]

Une métrique de qualité du logiciel est définie comme une échelle quantitative et une méthode pouvant être employées pour déterminer la valeur que prend un attribut d'une entité d'un produit logiciel spécifique.

Depuis leur apparition dans les années 70, les métriques automatiquement calculables sont devenues un outil important pour évaluer des attributs de logiciel et des activités logicielles connexes [5].

2. Historique

L'idée de créer et de définir des métriques du logiciel a commencé dans les années 1960 et 1970 et a été principalement développée dans les années 1980 et 1990.

En 1955, John Backus a proposé la métrique logicielle LOC (Line Of Code) pour analyser la taille d'un compilateur FORTRAN.

En 1971, la première tentative d'utiliser des métriques pour la prédiction de la qualité des logiciels en proposant un modèle basé sur la régression pour mesurer la densité des défauts dans un module.

En 1974, RW Wolverton a proposé l'utilisation des métriques basée sur la métrique LOC pour quantifier la productivité du programmeur.

En 1976, une métrique de complexité cyclomatique a été proposée par Thomas McCabe. Cette mesure tente de quantifier la complexité d'un programme en mesurant le nombre de chemin linéairement indépendants contenus dans le code source.

En 1977, Maurice Halstead a introduit un ensemble de métriques visant la complexité d'un programme ; en particulier la complexité de calcul. Ces métriques sont basées sur le nombre d'opérateurs et d'opérandes contenus dans le code source d'un programme.

En 1980, plusieurs mesures de complexité ont été introduites, y compris la métrique du diagramme de Ruston, qui décrit l'organigramme d'un programme à l'aide des éléments et de la structure sous-jacente de l'organigramme.

En 1981, Harison a introduit des métriques pour déterminer le niveau d'imbrication des graphes de flux.

En 1982, Troy a défini un ensemble de métriques dans le but de quantifier la modularité, la taille, la complexité, la cohésion et le couplage d'un programme.

En 1987, accordement de la programmation orientée objet dans le domaine de la mesure des logiciels.

En 1988, Rocacher a réalisé les premières recherches sur la mesure des conceptions orientées objets en établissant les métriques logicielles pour le langage de programmation orientée objet Smalltalk.

En 1990, la programmation orientée objet est devenue la méthode de programmation la plus répandue, l'accent est également changé dans la mesure des logiciels vers les programmes orientés objet. Plusieurs métriques orientée objet ont été ajoutées.

En 1992, le Standard IEEE décrit les méthodes pour établir les spécifications de la qualité ainsi que l'identification, l'analyse et la validation des métriques de la qualité.

En 1994, Chidamber et Kemerer ont introduit un ensemble de métriques orientées objet, connu sous le nom des métriques CK. Ces métrique sont largement utilisées pour quantifier les approches de conception orientées objet.

Entre 1991 et 2008, la métrique point de fonction peut être utilisée pour une variété de fins utiles économiques, y compris dans des études de la production du logiciel, des études de l'utilisation du logiciel et enfin dans des études de la qualité logicielle [6].

3. Intérêt

La mesure est une activité d'ingénierie qui permet d'obtenir une information quantitative sur les processus d'ingénierie ou les systèmes en cours de développement, cette activité de mesure est menée à la fois pour améliorer notre connaissance de la nature et des systèmes, mais aussi pour prédire une ou plusieurs caractéristiques finales du système développé [5].

D'après Zuse, Fenton et Henderson, la mesure du logiciel est essentiellement prédictive c'est-à-dire que le but principal de la mesure en phase amont du développement logiciel soit la prédiction de la qualité finale du logiciel produit.

Par conséquent la mesure tôt dans le cycle de développement permet aux architectes et aux managers d'estimer les coûts, d'identifier les risques et les défauts, de valider des propriétés et de suivre une démarche d'assurance qualité dès le début du développement.

Comme **Oman et Pfleeger** ont montré qu'il y a six objectifs pour mesurer.

Ces objectifs sont :

- mesurer pour la compréhension,
- mesurer pour l'expérimentation,
- mesurer pour le contrôle de projet,
- mesurer pour l'amélioration du processus,
- mesurer pour l'amélioration du produit
- et mesurer pour la prédiction.

D'après **Fenton**, les métriques logicielles peuvent être utilisées pour :

- **Estimation du coût et de l'effort** : pour l'estimation du coût et de l'effort pour les projets logiciels plusieurs modèles ont été proposés. Par exemple le modèle de Putnam SLIM, le modèle d'Albecht Points de fonction le modèle de Boehn COCOMO.
- **Les mesures de la productivité et modèles** : les besoins urgents de la gestion ont également donné lieu à de nombreuses tentatives de définir des mesures et des modèles pour évaluer la productivité du personnel au cours des différents processus de développement du logiciel et dans différents environnements.

- **La collecte de données** : la collecte de données est devenue une discipline en soi, où des spécialistes travaillent à s'assurer que les mesures soient définies sans ambiguïté, que la collecte est cohérente et complète, et que l'intégrité des données soit préservée.
- **Les modèles de qualité et mesures** : la productivité ne peut être considérée isolément. Sans une évaluation d'accompagnement de la qualité du produit, la vitesse de production est dénuée de sens. Cette observation a conduit les ingénieurs logiciels à développer des modèles de qualité dont les mesures peuvent être combinées avec celles des modèles de productivité.
- **Les modèles de fiabilité** : les modèles de fiabilité sont des modèles mathématiques qui permettent d'évaluer les mesures caractérisant la fiabilité du logiciel en évolution, à l'aide d'expressions relativement complexes.
- **L'évaluation des performances et modèles** : la performance est un autre aspect de la qualité, son évaluation inclut des caractéristiques de performance du système.
- **Métriques de complexité et des structures** : Métriques de complexité et des structures permettent de prédire quelles parties du système logiciel sont susceptibles d'être moins fiables, plus difficiles à tester, ou nécessitent plus de maintenance que les autres avant même que le système ne soit terminé.
- **La gestion par les métriques** : pour nous aider à décider si le projet est sur la bonne voie, elles doivent être présentées d'une manière qui indique à la fois au client et au développeur l'avancement du projet.
- **L'évaluation des méthodes et outils** : De nombreux organismes réalisent des expériences, mènent des études et des sondages pour les aider à décider si une méthode ou un outil est susceptible de faire la différence dans des situations spécifiques.
- **Maturité et évaluation des aptitudes** : Il peut être utilisé comme référentiel formel lors d'audits de fournisseurs, pour juger de leur capacité de réaliser un certain projet informatique [6].

4. Les métriques

Maintenant que nous avons défini les métriques et leurs utilités dans le développement du logiciel, nous présentons quelques exemples de métriques.

4.1. Les métriques logicielles

Fenton a classé les métriques logicielles par catégorie:

- **Les métriques de processus** : Ce sont des séries d'activités reliées au développement du logiciel. La quantification et la prédiction des attributs dans ce domaine (comme le temps, l'effort et le coût) sont particulièrement d'intérêt pour les gestionnaires et les chefs d'équipe.
- **Les métriques de ressource** : mesurent les ressources nécessaires à chacun des activités du processus de développement, telles que le personnel, logiciels et matériels. A cet égard, le personnel est requis pour compléter un processus de développement. Les attributs d'intérêt pour cette entité sont par exemple le nombre d'ingénieurs logiciels impliqués, leurs compétences et leurs performances.
- **Les métriques de produit** : détiennent les entités logicielles qui résultent du processus d'activité.

4.1.1. Les métriques de produit

Parmi les métriques de produit, on distingue les métriques traditionnelles et les métriques orientées objet.

- ❖ **les métriques traditionnelles** : elles se divisent en deux groupes : les métriques mesurant la taille et la complexité, et les métriques mesurant la structure du logiciel.

4.1.1.1. Les métriques mesurant taille et complexité

Les plus connues sont les métriques de ligne de code ainsi que les métriques de Halstead.

Les métriques de ligne de code : Pour quantifier la complexité d'un logiciel, les mesures les plus utilisées sont les lignes de code (LOC acronyme de « lines of code ») puisqu'elles sont simples, faciles à comprendre et à compter. Cependant ces mesures ne prennent pas en compte le contenu d'intelligence et la disposition du code. On peut distinguer les types de métriques de lignes de code suivants :

- **LOCphy**: nombre de lignes physiques (total des lignes des fichiers source) ;
- **LOCpro**: nombre de lignes de programme (déclarations, définitions, directives, et code ;
- **LOCcom**: nombre de lignes de commentaire ;
- **LOCbl**: nombre de lignes vides (en anglais number of blank lines) [7].

les métriques de Halstead : Les métriques de complexité Halstead ont été développées par le Maurice Halstead comme un moyen de déterminer une mesure quantitative de complexité directement des opérateurs et opérands dans le module afin de mesurer la complexité d'un module de programme depuis le code source. Du fait qu'elles soient appliquées au code, elles sont plus souvent utilisées comme une métrique de maintenance.

Il est évident que les mesures Halstead sont aussi utiles durant le développement, pour accéder à la qualité du code dans les applications denses. Les mesures de Halstead devraient être utilisées durant le développement du code, afin de suivre les tendances de complexité.

Les métriques Halstead sont classées en trois groupes:

Les mesures de base : sont des mesures qui comprennent-le :

- Nombre total des opérateurs uniques (n1)
- Nombre total des opérateurs (N1)
- Nombre total des opérands uniques (n2)
- Nombre total des opérands (N2)

Sur la base de ces chiffres on calcule :

- La Longueur du programme (N) : $N = N1 + N2$.
- La Taille du vocabulaire (n) : $n = n1 + n2$

Les mesures dérivées : ces mesures contiennent

- **Le volume** : on obtient le Volume du Programme (V) en multipliant la taille du vocabulaire par le logarithme 2 : $V = n * \log_2(n)$
- **La difficulté (D)** : la difficulté d'un programme est proportionnel au nombre d'opérateurs unique (n1) dans le programme et dépend également du nombre total d'opérands (N2) et du nombre d'opérands uniques (n2). Si les mêmes opérands sont utilisés plusieurs fois dans le programme, il est plus enclin aux erreurs. $D = (n1 / 2) * (N2 / n2)$
- **Le Niveau de programme (L)** est l'inverse du Niveau de difficulté. Un programme de bas niveau est plus enclin aux erreurs qu'un programme de haut niveau. $L = 1 / D$
- **L'Effort à l'implémentation (E)** est proportionnel au volume (V) et au niveau de difficulté (D). Cette métrique est obtenu par la formule suivante : $E = V * D$
- **Une approximation pour le Temps pour implémenter (T)** un programme en secondes est donnée par : $T = E / 18$
- **Le « nombre de bugs fournis »** est calculé selon la formule suivante:

$B = (E (2/3)) / 3000$. Cette valeur donne une indication pour le nombre d'erreurs qui devrait être trouvé lors du test de logiciel.

Les lignes de mesures de code : Les lignes de mesures de code comprennent des lignes vides, des lignes de code, et les lignes de commentaire [8].

4.1.1.2. Les métriques mesurant la structure du logiciel

Comme la complexité cyclomatique de McCabe se basent sur des organigrammes de traitement ou des structures de classe.

La métrique McCabe : c'est la métrique la plus utilisée après les lignes de code. Elle indique le nombre de chemins linéaires indépendants dans un module de programme et représente finalement la complexité des flux de données. Elle correspond au nombre de branches conditionnelles dans l'organigramme d'un programme.

Le nombre cyclomatique évalue le nombre de chemins d'exécution dans la fonction et ainsi donne une indication sur l'effort nécessaire pour les tests du logiciel.

Plus le nombre cyclomatique est grand, plus il y aura de chemins d'exécution dans la fonction, et plus elle sera difficile à comprendre et à tester.

Le Nombre cyclomatique de Mc Cabe $\rightarrow C = a - n + 2p$

Avec

- a = nombre d'arcs du graphe de contrôle
- n = nombre de nœuds du graphe de contrôle
- p = nombre de composantes connexes (1 le plus souvent)

Exemple

Ici, $n = 8$, $a = 11$ et $p = 1$
Donc $C = 11 - 8 + 2 = 5$

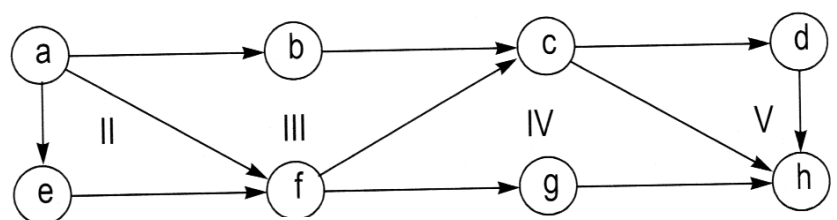


Figure 1: Graphe de mesure de complexité d'une classe

Les métriques McCabe sont une collection de quatre paramètres de logiciel :

- **Complexité cyclomatique :** ou $v(G)$ mesure le nombre de chemins linéairement indépendant.

- **Complexité essentielle** : ou $ev(G)$ est la mesure dans laquelle un flowgraph (un flowgraph est un graphe où chaque nœud correspond à une instruction de programme et chaque arc indique le flux de contrôle d'une déclaration à l'autre) peut être "réduite".
- **Complexité de la conception** : ou $iv(G)$ est la complexité cyclomatique de la réduction de flowgraph d'un module.
- **Lignes de Code (LOC)** : Le nombre total de lignes de code. Les lignes blanches et les commentaires ne sont pas comptabilisés.
- ❖ **les métriques orientées objet** : elles prennent en considération les relations (agrégation, association, généralisation, dépendance, composition, réalisation) entre éléments de programme (classes, méthodes).

5.2. Les métriques orientées objets

Les métriques Chidamber et kemerer sont un ensemble de métriques pour la conception orientée objet, conçu pour permettre d'effectuer une évaluation des classes d'un système. CK peut être utilisé à diverses fins d'analyse, par exemple la prédiction de défaut de logiciel.

Chidamber et Kemerer proposent six métriques pour les classes des logiciels orientés-objets :

- **WMC - Weighted Methods per Class (poids des méthodes par classe)**

Est simplement la somme de la complexité de ses méthodes. En tant que mesure de la complexité, nous pouvons utiliser la complexité cyclomatique, ou nous pouvons attribuer arbitrairement une valeur de complexité de 1 à chaque méthode. Le programme de CKJM (Chidamber et Kemerer Java Metric) attribue une valeur de complexité de 1 à chaque méthode, et donc la valeur de la WMC est égal au nombre de méthodes de la classe.

- **DIT - Depth of Inheritance Tree (profondeur d'arbre d'héritage)**

Le DIT pour une classe est la longueur du trajet à partir de la classe à la racine de la hiérarchie d'héritage.

- **NOC – Number Of Children (nombre d'enfants)**

C'est une métrique qui mesure le nombre de descendants immédiats de la classe.

- **CBO - Coupling Between Object classes (couplage entre objets d'une classe)**

C'est une métrique qui représente le nombre de classes couplées à une classe donnée. Ce couplage peut se produire par les appels de méthode, accès aux champs, l'héritage, les arguments, les types et les exceptions retournées.

- **RFC - Response For a Class (nombre de réponse pour chaque classe)**

Cette métrique mesure le nombre de différentes méthodes qui peuvent être exécutées quand un objet de cette classe reçoit un message (lorsqu'une méthode est invoquée pour cet objet).

- **LCOM - Lack of Cohesion in Methods (manque de cohésion de méthode)**

Cette métrique compte les ensembles de méthodes dans une classe qui ne sont pas liés par le partage d'une partie des champs de la classe [10].

Les métriques proposées par Li et Henry

Cette proposition reprend cinq des six métriques de Chidamber et Kemerer, en y ajoutant :

- **MPC -Message Passing Coupling (couplage de passage des messages)**

Nombre de messages envoyés par une classe en direction des autres classes du système (nombre de méthodes invoquées).

- **DAC - Data Abstraction Coupling (couplage des données abstraits)**

Nombre de types de données abstraites définies dans une classe (classes dont la définition est incluse dans la définition d'une autre classe), ou attribut d'une classe qui est une autre classe.

- **NOM - Number Of local Methods (Nombre des methods locales)**

Nombre de méthodes localement définies dans une classe (hors méthodes héritées).

- **SIZE1**

Nombre d'instructions dans l'implémentation d'une classe.

- **SIZE2**

Cumul du nombre d'attributs et du nombre de méthodes locales (NOM) d'une classe. [10]

Les métriques MOOD (Metrics for Object Oriented Design)

Ces métriques sont proposées par Abreu, GouUio et Esteves, en 1995, Ensemble de métriques pour mesurer les attributs des propriétés suivantes [9]:

- **Encapsulation :**

- ❖ **MHF (Method Hiding Factor)** : mesure en général le masquage moyen des méthodes. Une méthode M est visible pour les objets d'une classe C (et par extension visible pour la classe C) si ceux-ci peuvent l'appeler.

$$MHF = \frac{\sum_{i=1}^{TC} M_h(C_i)}{\sum_{i=1}^{TC} M_d(C_i)}$$

Avec

$M_d(C_i)$ Le nombre de méthodes déclarées dans une classe C_i

$M_h(C_i)$ Le nombre de méthodes cachées

TC le nombre total de classes.

- ❖ **AHF (Attribute Hiding Factor)** : définition analogue à MHF pour les attributs.

$$AHF = \frac{\sum_{i=1}^{TC} A_h(C_i)}{\sum_{i=1}^{TC} A_d(C_i)}$$

Avec

$A_d(C_i)$ Le nombre d'attributs déclarés dans une classe C_i

$A_h(C_i)$ Le nombre d'attributs cachés

- **Héritage :**

- ❖ **MIF (Method Inheritance Factor)** : correspond globalement au pourcentage de méthodes héritées (non définies localement).

$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)}$$

Avec

$M_i(C_i)$ Le nombre de méthodes héritées (et non surchargées) de C_i

$M_a(C_i)$ Le nombre de méthodes qui peuvent être appelées depuis la classe i

- ❖ **AIF (Attribute Inheritance Factor)** comme MIF mais pour les attributs.

$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)}$$

Avec

$A_i(C_i)$ Le nombre d'attributs hérités de C_i

$A_a(C_i)$ Le nombre d'attributs auxquels C_i peut accéder

- **Couplage**

- ❖ **CF (Coupling Factor) :** Mesure le couplage entre les classes sans prendre en compte celui dû à l'héritage.

$$CF = \frac{\sum_{i=1}^{TC} \sum_{j=1}^{TC} \text{client}(C_i, C_j)}{TC^2 - TC}$$

Avec

$\text{client}(C_i; C_j) = 1$ si la classe i a une relation avec la classe j , et 0 sinon

- **Polymorphisme :**

- ❖ **PF (Polymorphic Factor) :** pourcentage de méthodes polymorphes par rapport au nombre total de méthodes potentiellement polymorphes.

$$PF = \frac{\sum_{i=1}^{TC} M_o(C_i)}{\sum_{i=1}^{TC} M_n(C_i) * DC(C_i)}$$

Avec

$M_o(C_i)$ Le nombre de méthodes surchargées dans la classe i

$M_n(C_i)$ Le nombre de nouvelles méthodes dans la classe i

$DC(C_i)$ le nombre de descendants de la classe i

III. Conclusion

Nous avons présenté dans ce chapitre quelques exemples de métriques dans différents contextes, de plus, Nous avons voulu mettre l'accent sur la nécessité de la métrique pour quantifier les analyses et augmenter la qualité.

Chapitre3 : Métriques du diagramme de classes

I. Introduction

Dans ce chapitre nous allons présenter l'importance du langage de modélisation unifié (UML) et nous mettons l'accent sur son diagramme de classe qui est un élément crucial dans la phase de conception d'un logiciel, ainsi que les métriques utilisés dans ce diagramme pour la prédiction des défauts afin de renforcer la qualité logiciel qui sera implémenté.

I. Le langage de modélisation unifié

1. Définition

Le langage de modélisation unifié, de l'anglais Unified Modeling Language (UML), est un langage de modélisation graphique à base de pictogrammes conçu pour fournir une méthode normalisée pour visualiser la conception d'un système. Il est couramment utilisé en développement logiciel et en conception orientée objet [11].

2. L'intérêt

UML est utilisé pour spécifier, visualiser, modifier et construire les documents nécessaires au bon développement d'un logiciel orienté objet. UML offre un standard de modélisation, pour représenter l'architecture logicielle [11].

II. Le diagramme de classes

1. Définition

Le diagramme de classes est un schéma utilisé en génie logiciel pour présenter les classes et les interfaces des systèmes ainsi que les différentes relations entre celles-ci [12].

2. L'intérêt

Les diagrammes de classes vous aident à comprendre la structure des classes des projets écrits par d'autres personnes. Vous pouvez les utiliser pour personnaliser, partager, présenter des informations de projet et décrire les classes d'une application et leurs relations statiques [12].

3. Les composants du diagramme de classes

- **Les paquetages** : Mécanisme de partitionnement des modèles et de regroupement des éléments de modélisation. Chaque paquetage peut contenir un ensemble de diagrammes et/ou de paquetages.

Chaque élément d'un paquetage possède un nom unique dans ce paquetage.

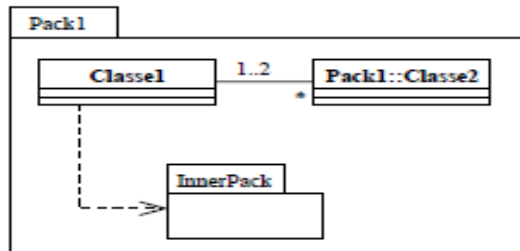


Figure 2:Présentation graphique d'un paquetage d'un diagramme de classe

- **les classes** : ce sont une description d'un ensemble d'objet partageant les mêmes attributs, opérations, méthodes, relations et sémantiques.

Les classes sont décrit par :

- ❖ **Un nom** : Obligatoire, commence par une majuscule, en gras.
- ❖ **Des attributs** : propriétés définies par un nom, un type et éventuellement une valeur initiale.
- ❖ **Des opérations** : spécifications du comportement des instances de la classe.

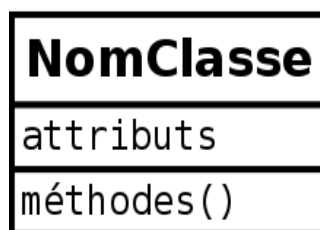


Figure 3:La structure générale d'une classe

UML définit trois niveaux de visibilité :

- ❖ **public** : l'élément est visible pour tous les clients/utilisateurs de la classe, noté par +.
- ❖ **protégé** : l'élément est visible pour les sous-classes de la classe, noté par #.
- ❖ **privé** : l'élément est visible pour la classe seule, noté par -.

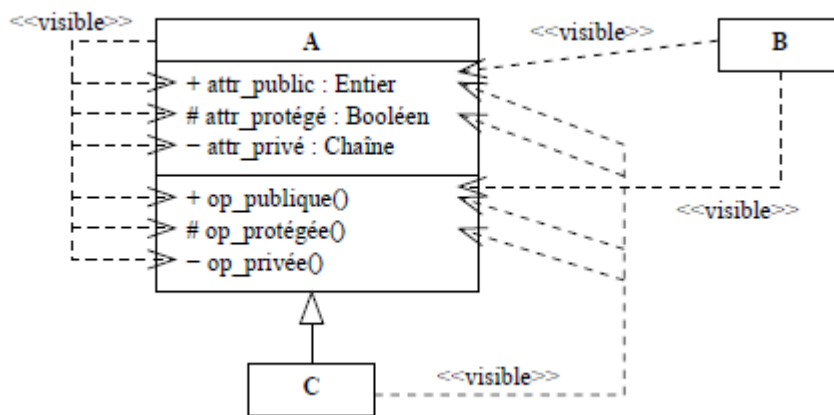


Figure 4:Exemple des niveaux de visibilité

• Les relations :

- ❖ **L'héritage** : relation entre une classe plus générale et une classe plus spécifique (signifie "est un" ou "est une sorte de").

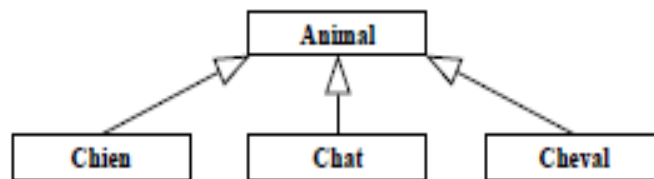


Figure 5:Exemple d'héritage

- ❖ **Association** : c'est la relation entre au moins deux classes qui entraînent des connexions entre leurs instances. Elle peut être binaire, dans ce cas elle est représentée par un simple trait, ou n-aire, les classes sont reliées à un losange par des traits simples.

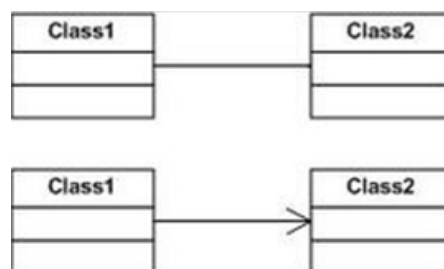


Figure 6:Exemple d'une relation d'association

- ❖ **Agrégation** : Association avec relation de subordination, représentée par un trait reliant les deux classes et dont l'origine se distingue de l'autre extrémité (la classe subordonnée) par un losange. Une des classes "regroupe" d'autres classes. On peut dire que l'objet T utilise une instance de la classe T'.

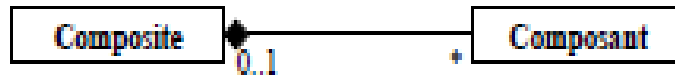


Figure 7:Exemple d'une relation d'agrégation

- ❖ **Composition** : Agrégation avec cycle de vie dépendant (on dit que la classe composée est détruite lorsque la classe mère disparaît).

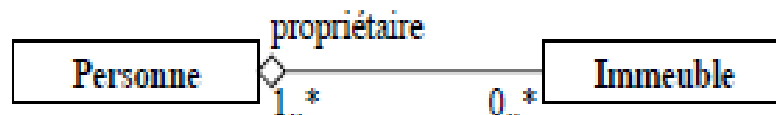


Figure 8:Exemple d'une relation de composition

- ❖ **Dépendance** : Implique qu'une ou plusieurs méthodes reçoivent un objet d'un type d'une autre classe. Il n'y a pas de liaison en ce qui concerne la destruction d'objets mais une dépendance est quand même là ! Elle est symbolisée par une flèche en pointillés, dont son extrémité possède trois traits qui se coupent en un même point. [12]

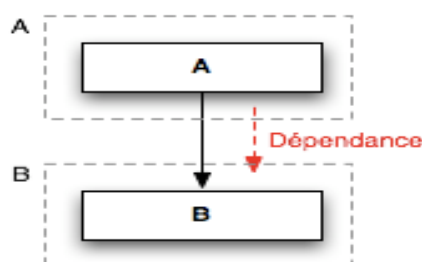


Figure 9: Exemple d'une relation de dépendance

III. Les métriques du diagramme de classe

Dans cette partie nous introduisons un ensemble de métriques sélectionnées à partir des diagrammes de classes lors de l'étape initiale d'un développement orienté-objet.

1. Les métriques de Chidamber et Kermer

Nous avons cité dans le chapitre précédent les six métriques de conception OO proposé par Chidamber et Kermer, parmi ces métriques : le WMC (Weighted Method per Class), DIT (Depth Inheritance Tree) et NOC (Number of Children) sont seulement les deux mesures qui peuvent être appliquées à un diagramme de classes UML.

Basili a mis ces métriques en cours de validation empirique, de conclure que plus la DIT et NOC ont de grandes valeurs, plus la probabilité de détection de défaut est importante.

2. Les métriques MOOD (Metrics for Object Oriented Design)

Parmi les six métriques de MOOD qui peuvent être appliqués à des éléments d'un diagramme de classes sont :

- **MHF (Method Hiding Factor)**
- **AHF (Attribute Hiding Factor)**
- **MIF (Method Inheritance Factor)**
- **AIF (Attribute Inheritance Factor)**
- **PF (Polymorphism Factor)**

Pour la métrique MHF, Brito e Abreu et Melo ont démontré empiriquement que lorsque la valeur de MHF augmente, la densité de défauts et les efforts nécessaires pour y remédier devraient diminuer.

3. Les métriques de Li et Henry

Li et Henry ont proposé 5 métriques qui permettent de mesurer des attributs internes tels que le couplage, la complexité et la taille. Parmi ces métriques nous pouvons sélectionner 3 métriques qui peuvent être mesurées à partir du diagramme de classe.

- **DAC (Data Abstracting Coupling)**
- **NOM (Number Of local Methods)**
- **SIZE2**

4. Métrique de Lorenz et Kidd

Les métriques de Lorenz et kid sont définies pour mesurer les caractéristiques statiques de la conception de logiciels. Lorenz et Kidd ont classé leurs mesures ainsi [13] :

❖ Des métriques de taille des classes

- **PIM (Public Instance Method)** : est définie comme le nombre total de méthodes d'instance publique dans une classe.
- **NIM (Number Instance Method)** : compte toutes les méthodes publiques, protégées et privées définies pour les instances de classes.
- **NIV (Number Instance Variable)** : est défini comme le nombre total de variables d'instance comprennent des variables privées et protégées à la disposition des instances.
- **NCM (Number of Class Methods)** : est défini comme le nombre total de méthodes de classes dans une classe. Une méthode de classe est une méthode qui est globale à ses instances.
- **NCV (Number of Class Variables)** : est défini comme le nombre total de variables de classes dans une classe.

❖ Des métriques d'héritage de classe

- **NMO (Number of Methods Overridden)** : défini comme le nombre total des méthodes redéfinies par une sous-classe.
- **NMI (Number of Methods Inherited)** : est défini comme le nombre total de méthode héritée par une sous-classe.
- **NMA (Number of Methods Added)** : est défini comme étant le nombre total des méthodes définies dans une sous-classe.
- **SIX (Specialisation Index)** : l'indice de spécialisation pour chaque classe est défini ainsi:

$$\frac{\text{nombre des méthodes redéfinies} * \text{le niveau de la hiéarchie}}{\text{nombre total des méthodes}}$$

❖ Des métriques des classes internes

- **APPM (Average Parameters Per Method)** : la moyenne des paramètres par méthode :

$$\frac{\text{nombre total des paramètres des méthodes}}{\text{nombre total des méthodes}}$$

5. Les métriques de Briand

Ces métriques sont définies au niveau de la classe, et mesurent l'interaction entre classes [13].

Tableau 1:Tableau résume les différentes métriques de Briand

Nom des métriques	Définition
ACAIC OCAIC DCAEC OCAEC ACMIC OCMIC DCMEC OCMEC	<p>Les acronymes de ces mesures indiquent quelles interactions sont comptabilisées:</p> <ul style="list-style-type: none">○ La première lettre indique la relation (A: couplage entre les classes ancêtres, D: Descendants, O: autres, c'est à dire qu'aucun des autres relations).○ Les deux lettres suivantes indiquent le type d'interaction:<ul style="list-style-type: none">• CA: Il existe une interaction classe-attribut entre les classes C et D, si C a un attribut de type D.• CM: Il existe une interaction classe- Méthode entre les classes C et D, si la classe C a une méthode avec paramètre de type classe D.○ Les deux dernières lettres indiquent le lieu de l'impact:<ul style="list-style-type: none">• IC: couplage d'importation, la mesure compte pour une classe C toutes les interactions où C utilise une autre classe.• CE: couplage d'exportation: les interactions de comptage où la classe D est la classe utilisée.

6. Métriques de Harrison

Les auteurs ont proposé la métrique du nombre d'associations (NAS-Number Association-), qui est défini comme le nombre d'associations de chaque classe, compté par le nombre de lignes d'association émanant d'une classe dans un diagramme de classes. Cette métrique mesure le couplage inter-classe [13].

7. Métriques de Genero

Ils ont été définis pour mesurer la complexité du diagramme de classe, en raison de l'utilisation de différents types de relations, telles que les associations, les généralisations,

d'agrégation et dépendances, en relation avec leur impact sur la qualité externe des attributs tels que la maintenabilité du diagramme de classes. Ces mesures ont été regroupées en [13] :

❖ Métriques appliqués aux classes

- **NAssoc (Number of Association per Class):** définie comme le nombre total d'associations d'une classe avec d'autres classes, ou avec lui-même.
- **HAgg (Height of a class within an Aggregation hierarchy) :** est défini comme étant la longueur du chemin le plus long à partir des classes feuilles.
- **NODP (Number of Direct Parts) :** est défini comme le nombre total de "partie directe" des classes qui composent une classe composite.
- **NP (Number of Parts):** est défini comme le nombre de classes «partielles» (directes et indirectes) d'une classe «tout».
- **NW (Number of Wholes):** est défini comme le nombre de classes «toute» (directes ou indirectes) d'une classe "partielle".
- **MAgg (Multiple Aggregation):** est défini comme le nombre de classes directes "entières" qui ont une classe dans une hiérarchie d'agrégation.
- **NDepIn (Number of Dependencies In) :** est défini comme le nombre de classes qui dépendent d'une classe donnée.
- **NDepOut (Number of Dependencies Out) :** est défini comme le nombre de classes à laquelle une classe donnée dépend.

❖ Métriques de diagramme de classe:

- **NAssoc (Number of Association) :** est définie comme le nombre total d'associations dans un diagramme de classes. C'est une généralisation de la métrique NAS (nombre d'associations) au niveau de diagramme de classe.
- **NAgg (Number of Aggregation):** Le numéro de l'agrégation métrique est défini comme le nombre total de relations d'agrégation dans un diagramme de classes
- **NDep (Number of Dependencies) :** est définie comme le nombre total de relations de dépendance dans un diagramme de classe.
- **NGen (Number of Generalization) :** Le Nombre de généralisation métrique est définie comme le nombre total de relations de généralisation dans un diagramme de classes

- **NGenH (Number of Generalization Hierarchies)** : est définie comme le nombre total des hiérarchies de généralisation dans un diagramme de classe.
- **NAggH (Number of Aggregation Hierarchies)** : est définie comme le nombre total des hiérarchies d'agrégation dans un diagramme de classe.
- **MaxDIT (Maximum DIT)** : est définie comme le maximum entre la valeur de DIT obtenu pour chaque classe du schéma de classe.
- **MaxHAgg (Maximum HAgg)** : est définie comme le maximum entre la valeur Hagg obtenu pour chaque classe du diagramme de classe.

Ces paramètres ont été validés à l'aide d'une approche basée sur les propriétés, visant à les classer comme les métriques de la complexité, de la taille, de la longueur, du couplage ou de la cohésion.

Tableau 2:La validation théorique des métriques utilisant Briand et al.

	taille	complexité	longueur	couplage
Métriques de la classe-diagramme portée	NAggH NGenH	NAssoc NDep NAgg NGen	MaxHAgg MaxDIT	
Métrique de la classe-portée	NDP NP NW		HAgg	NAssoc NDepIN NDepOUT

8. Métriques de Bansiya

Nous présentons les métriques définies par Bansiya et Davis, que nous pouvons l'appliquer au niveau du diagramme de classe.

- **DAM (Data Access Metric)**: La métrique d'accès aux données est le rapport entre le nombre d'attribues privé (protégé) et nombre total d'attributs déclarés dans la classe.
- **DCC (Direct Class Coupling metric)**: La mesure du couplage direct d'une classe est le nombre différent de classes directement liée à une classe. La métrique inclut des classes qui sont directement liées par les déclarations d'attributs et de passage (paramètres) dans les méthodes de messages.
- **CAMC (Cohesion Among Methods of Class metric)** : La métrique de cohésion entre les méthodes d'une classe, calcule le degré de parenté entre les méthodes d'une classe basée sur la liste des paramètres de méthodes. La métrique est calculée à l'aide de la sommation de l'intersection des paramètres d'une méthode avec l'indépendant maximale de tous les types de paramètres dans la classe.

- **MOA (Measure of Aggregation metric)** : La métrique de la mesure d'agrégation est le nombre de déclarations de données dont les types sont des classes définies par l'utilisateur.
- **MFA (Measure of Functional Abstraction metric)** : La métrique de la mesure de l'Abstraction fonctionnelle est le rapport entre le nombre de méthodes héritées par une classe et le nombre total de méthodes accessibles par les méthodes membres de la classe.

IV. Conclusion

Dans ce chapitre, nous avons présenté les métriques orientées objet qui mesurent la complexité des diagrammes de classes UML obtenus dans les phases initiales du cycle de vie du développement d'un logiciel orienté objet.

Chapitre4 : Aide à la décision

I. Introduction

Comme tous les domaines, le génie logiciel a besoin d'accès et d'analyse préalable de donnée pour appliquer des techniques d'extraction des connaissances, afin de prendre les bonnes décisions.

II. L'aide à la décision

1. Définition

L'aide à la décision est l'ensemble des techniques permettant, pour une personne donnée, d'opter pour la meilleure prise de décision possible [14]. Elle peut être aussi définie comme la prise en compte de l'expérience, des données, et des connaissances spécifiques d'un problème, l'analyse et l'intégration de cette information pour produire un résultat aidant les décideurs [15].

2. L'historique

- Au XXe siècle, des outils mathématiques sont introduits. Ces modèles et leurs algorithmes s'appuient sur des concepts et théories tels que les probabilités, l'analyse de la décision, la théorie des graphes, ou encore la recherche opérationnelle.
- Rapidement, des systèmes informatiques d'aide à la décision sont apparus et ont pris une place croissante dans certains processus de décision, au point parfois de remplacer l'Homme par des processus automatiques.
- Les SIG (Systèmes d'Information Géographiques) se sont beaucoup développés depuis les années 1970, avec l'avantage de présenter visuellement et de manière cartographique certains éléments d'aide à la décision. Avec l'informatique sont également arrivés des Systèmes Interactifs d'Aide à la Décision (SIAD).
- Le premier système interactif d'aide à la décision a été publié dès 1971.
- Des agences et bureaux d'études se sont spécialisés dans le conseil et l'aide à la décision auprès des banques, entreprises, gouvernements, collectivités (via les systèmes d'assistance à la maîtrise d'ouvrage, par exemple).
- Par la suite s'est installée la Business Intelligence, et l'arrivée de nouvelles méthodes telles que OLAP, et les data warehouse.

- La gouvernance multi-niveaux et les processus de codécision qui tendent à se développer en Europe, ou dans le monde sous l'égide de l'ONU (Organisation des Nations Unies), exigent des outils transparents, souples et prenant en compte les différentes échelles de contextes sociologiques, économiques, environnementaux et de responsabilité. Dans ce domaine et celui de la démocratie participative la notion de "porté à connaissance" et d'accès à l'information, en particulier à la donnée publique ont pris une importance croissante.
- Avec l'Internet, on a vu apparaître de nouveaux moyens de consultation et d'expression ainsi que des outils de travail collaboratif permettant de nouveaux types de processus de décision et d'aide à la décision. Il y a également l'apparition de Big data pour le traitement de très grandes bases de données [14].

3. Domaines d'application

L'aide à la décision est principalement utilisée dans des domaines importants tels que : la finance et la banque, philosophie, psychologie, sociologie, recherche opérationnelle, l'informatique ou même la politique [14].

4. Rôle d'aide à la décision

Il est devenu essentiel de bénéficier d'outils « simples » permettant de vérifier et d'analyser rapidement les informations afin de pouvoir prendre la décision la plus adaptée à un instant donné et ce, sans nécessairement avoir des connaissances poussées en informatique. Les outils d'aide à la décision visent à répondre à ces problématiques.

Les méthodes d'aide à la décision permettent non seulement de fournir l'information, mais aussi de choisir parmi plusieurs solutions, en fonction de critères établis. Les outils d'aide à la décision aident le décideur à formuler un choix de façon plus transparente et plus robuste [14].

5. Intérêt d'outil d'aide à la décision

Les outils d'aide à la décision permettent d'apporter des réponses pertinentes à des problématiques diverses mettant en œuvre plusieurs choix possibles (implantation de sites industriels, stratégie de dépollution d'un lac, constitution de portefeuilles de valeurs, etc.), d'aider au diagnostic et, plus généralement, de faciliter la prise de décision stratégique ou opérationnelle en environnement imprécis et/ou incertain [14].

4.1.Des outils d'aide à la discision

Nous pouvons citer des outils d'aide à la discision tel que :

- **CDM DECISION TOOL** : un outil d'aide à la décision pour les projets carbonés (Porteurs de petits projets économes en CO2 dans les pays en développement).
- **le système BOS** (l'acronyme de son nom néerlandais, Beslis & Ondersteunend Systeem). Ce système d'aide à la décision détermine s'il est nécessaire de fermer les barrages lors d'une tempête en analysant les données recueillies aux stations et aux bouées météorologiques à proximité. C'est un système intelligent d'aide à la décision pour la gestion des barrages de protection contre les inondations. Le système BOS utilise ces données pour établir, toutes les dix minutes, une prévision de l'élévation du niveau d'eau pour les villes de Rotterdam et de Dordrecht. Lorsqu'il détecte un risque d'inondation, il met en œuvre une série de mesures de précaution et, si nécessaire, démarre de façon autonome le processus de fermeture des barrages de protection contre les ondes de tempête.
- **LG Nutrition Animale / Profil Ration** pour choisir la variété de maïs adaptée à la ration souhaitée,
- **LG Nutrition Animale / Profil Prairie** pour choisir les espèces et variétés fourragères,
- **LG Vision Maïs Fourrage** pour prévoir la date de récolte,
- **LG Vision Maïs Grain et Tournesol** pour piloter au mieux ces deux cultures.

4.2.Les avantages

Les outils d'aide à la décision aident à rendre la prise de décision robuste, consistante et reproductible. En effet, les outils d'aide à la décision fournissent des moyens de documenter les paramètres et les hypothèses utilisées dans l'analyse d'une décision particulière. Ils permettent d'améliorer la transparence du processus [15].

4.3.Les inconvénients

Même si de nombreux avantages sont mentionnés dans l'utilisation des outils d'aide à la décision, leur acceptation par toutes les parties prenantes est souvent difficile à atteindre. Si l'outil n'est pas compris, les personnes non impliquées dans l'utilisation de l'outil ne valideront pas les résultats. Il convient donc aux développeurs et aux utilisateurs de fournir une information claire et transparente sur l'outil afin d'éviter cette perception [15].

6. L'Informatique dans l'aide à la décision

Les progrès de l'informatique ont intégré l'aide à la décision, domaine visant à concevoir des outils informatiques (dont les logiciels experts) pour aider un décideur à analyser un problème ou une situation, et à lui fournir des solutions, éventuellement hiérarchisées sur la base des critères logiques qu'il aura sélectionnés.

La décision en entreprise résulte d'un processus toujours plus complexe : les données à prendre en compte sont toujours plus volumineuses et les enjeux si importants (humains, financiers) que l'outil informatique est devenu stratégique. Les systèmes décisionnels de traitement et de valorisation des données sont aujourd'hui bien implantés dans les entreprises. Ils s'enrichissent notamment à présent de puissantes méthodes analytiques, de prospection et d'optimisation.

Ainsi l'informatique décisionnelle désigne les moyens, les outils et les méthodes qui permettent de collecter, consolider, modéliser et restituer les données d'une entreprise en vue d'offrir une aide à la décision, et de permettre aux responsables d'une entreprise d'avoir une vue d'ensemble de l'activité traitée.

Les différentes applications de l'informatique dans la prise de décision classées dans leur ordre chronologique d'apparition :

- La première application de l'informatique apparaît avec les systèmes SIAD (Système Informatique d'Aide à la Décision). Il s'agit des systèmes informatiques intégrés, conçus spécialement pour la prise de décisions, et qui sont destinés plus particulièrement aux dirigeants d'entreprises.
- Apparaît par la suite l'intelligence artificielle, basée sur des formes de programmation originales cherchant généralement à simuler des réalités de manières virtuelles pour prendre des décisions suivant les résultats obtenus grâce à ces simulations. Les méthodes d'aide à la décision utilisant l'intelligence artificielle sont généralement basées sur les systèmes experts, les réseaux de neurones et les systèmes multi-agents par exemple.
- Plus récemment et en réponse à la constante évolution du volume de données stockées et exploitables, l'apparition de la Business Intelligence (BI) est venue donner un souffle nouveau aux méthodes et outils d'aide à la décision. Celle-ci est principalement basée sur l'extraction au sein d'importantes et nombreuses bases de données (autrement

appelées Entrepôts de données ou Data Warehouse). Cette fouille de données est également appelée DataMining.

III. Datamining

1. Définition

Le DataMining (l'exploration de données), connue aussi sous l'expression de fouille de données, forage de données, prospection de données, ou encore extraction de connaissances à partir de données, a pour objet l'extraction d'un savoir ou d'une connaissance à partir de grandes quantités de données, par des méthodes automatiques ou semi-automatiques [16].

2. Les domaines d'application

Voici une liste non exhaustive des applications possibles du datamining par secteur d'activités [17]:

- **Grande distribution et VPC (vente par correspondance)** : Analyse des comportements des consommateurs, recherche des similarités des consommateurs en fonction de critères géographiques ou socio-démographiques, prédiction des taux de réponse en marketing direct, vente croisée et activation sélective dans le domaine des cartes de fidélité, optimisation des réapprovisionnements.
- **Laboratoires pharmaceutiques** : Modélisation comportementale et prédiction de médicaments ou de visites, optimisation des plans d'action des visiteurs médicaux pour le lancement de nouvelles molécules, identification des meilleures thérapies pour différentes maladies.
- **Banques** : Modélisation prédictive des clients partants, détermination de pré-autorisations de crédit.
- **Assurance** : Modèles de sélection et de tarification, analyse des sinistres, recherche des critères explicatifs du risque ou de la fraude, prévision d'appel sur les plateformes d'assurance directe.
- **Aéronautique, automobile et industries** : Contrôle qualité et anticipation des défauts, prévision des ventes, dépouillement d'enquêtes de satisfaction.
- **Transport et voyagistes** : Optimisation des tournées, prédiction de carnets de commande, marketing relationnel dans le cadre de programmes de fidélité.

- **Télécommunications, eau, énergie** : Simulation de tarifs, détection de formes de consommations frauduleuses, classification des clients selon la forme de l'utilisation des services, prévision de ventes.

3. Les processus de Data Mining

Plus précisément, le datamining peut se décomposer en 8 étapes [17]:

1. Poser le problème
2. Rechercher des données
3. Sélectionner les données pertinentes
4. Nettoyer des données
5. Transformer les variables
6. Rechercher le modèle
7. Evaluer le résultat
8. Intégrer la connaissance

4. Les tâches de Data Mining

Le Data Mining n'est pas le remède miracle capable de résoudre toutes les difficultés ou besoins de l'entreprise. Cependant, une multitude de problèmes d'ordre intellectuel, économique ou commercial peuvent être regroupés, dans leur formalisation, dans l'une des tâches suivantes [18]:

4.1. Classification

La classification consiste à examiner des caractéristiques d'un élément nouvellement présenté afin de l'affecter à une classe d'un ensemble prédéfini.

La classification permet de créer des classes d'individus (terme à prendre dans son acception statistique). Celles-ci sont discrètes : homme / femme, oui / non, rouge / vert / bleu

Les techniques les plus appropriées à la classification sont :

- ❖ **Les arbres de décision,**
- ❖ Le raisonnement basé sur la mémoire,
- ❖ Eventuellement l'analyse des liens.

4.2. Estimation

Contrairement à la classification, le résultat d'une estimation permet d'obtenir une variable continue. Celle-ci est obtenue par une ou plusieurs fonctions combinant les données en entrée. Le résultat d'une estimation permet de procéder aux classifications grâce à un barème. Par exemple, on peut estimer le revenu d'un ménage selon divers critères (type de véhicule et nombre, profession ou catégorie socioprofessionnelle, type d'habitation, etc.). Il sera ensuite possible de définir des tranches de revenus pour classer les individus.

Un des intérêts de l'estimation est de pouvoir ordonner les résultats pour ne retenir si on le désire que les n meilleures valeurs. Cette technique sera souvent utilisée en marketing, combinée à d'autres, pour proposer des offres aux meilleurs clients potentiels. Enfin, il est facile de mesurer la position d'un élément dans sa classe si celui-ci a été estimé, ce qui peut être particulièrement important pour les cas limitrophes.

La technique la plus appropriée à l'estimation est : **le réseau de neurones.**

4.3. Prédiction

La prédiction ressemble à la classification et à l'estimation mais dans une échelle temporelle différente. Tout comme les tâches précédentes, elle s'appuie sur le passé et le présent mais son résultat se situe dans un futur généralement précisé. La seule méthode pour mesurer la qualité de la prédiction est d'attendre.

Les techniques les plus appropriées à la prédiction sont :

- ❖ Le raisonnement basé sur la mémoire
- ❖ **Les arbres de décision**
- ❖ **les réseaux de neurones**

4.4. Groupement par similitudes

Le regroupement par similitudes consiste à grouper les éléments qui vont naturellement ensembles. La technique la plus appropriée au regroupement par similitudes est l'analyse du panier de la ménagère.

4.5.Segmentation (ou clusterisation)

L'analyse des clusters consiste à segmenter une population hétérogène en sous populations homogènes. Contrairement à la classification, les sous populations ne sont pas préétablis. La technique la plus appropriée à la clusterisation est l'analyse des clusters

4.6.Description

C'est souvent l'une des premières tâches demandées à un outil de Data Mining. On lui demande de décrire les données d'une base complexe. Cela engendre souvent une exploitation supplémentaire en vue de fournir des explications. La technique la plus appropriée à la description est l'analyse du panier de la ménagère.

4.7. Optimisation

Pour résoudre de nombreux problèmes, il est courant pour chaque solution potentielle d'y associer une fonction d'évaluation. Le but de l'optimisation est de maximiser ou minimiser cette fonction. Quelques spécialistes considèrent que ce type de problème ne relève pas du Data Mining. La technique la plus appropriée à l'optimisation est le réseau de neurones.

IV. Les techniques les plus appropriées au Data Mining

1. Réseaux de neurones

1.1.Définition

Un réseau de neurones artificiels est un modèle de calcul dont la conception est très schématiquement inspirée du fonctionnement des neurones biologiques.

Un réseau neuronal est l'association, en un graphe plus ou moins complexe, d'objets élémentaires, les neurones formels. Les principaux réseaux se distinguent par l'organisation du graphe (en couches, complets. . .), c'est-à-dire leur architecture, son niveau de complexité (le nombre de neurones, présence ou non de boucles de rétroaction dans le réseau), par le type des neurones (leurs fonctions de transition ou d'activation) et enfin par l'objectif visé : apprentissage supervisé ou non, optimisation, systèmes dynamiques [19].

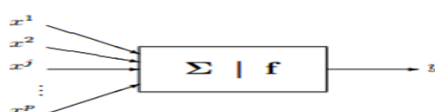


Figure 10:Représentation d'un neurone formel

1.2. Histoire

Fin de 1943, les neurologues McCulloch et Pitts ont présenté un modèle simplifié de neurone biologique : neurone formel

1949 : Hebb propose un mécanisme d'apprentissage

1958, Franck Rosenblatt a développé le modèle du perceptron et Windrow a présenté le modèle de l'ADALINE (ADaptive LINear Element) : le modèle de base des réseaux multicouches.

1969, Marvin Lee Minsky et Seymour Papert ont publié un ouvrage mettant en exergue les limitations du perceptron,

1972 : Kohonen présente ses travaux sur les mémoires associatives.

1982 : Hopfield démontre l'intérêt d'utiliser les réseaux récurrents pour la compréhension et la modélisation des fonctions de mémorisation.

1986 : Rumelhart popularise l'algorithme du gradient, conçue par Werbos, qui permet la rétropropagation et l'entraînement par couches cachées des réseaux multicouches.

1.3. Application

Les réseaux de neurones ont été beaucoup étudiés, et ont trouvé énormément d'application tels que:

Aérospatial : pilotage automatique, simulation du vol...

Défense : guidage de missiles et suivi de cible,

Electronique : prédiction de la séquence d'un code, vision machine, synthétiseur vocal,...

Finance : Prévision du coût de la vie

Télécommunication : Compression de données ...

1.4. Modèle biologique

- Les réseaux de neurones fortement inspirés par le système nerveux biologique.
- Les neurones reçoivent des signaux (impulsions électriques) par les dendrites et envoient l'information par les axones.
- Les contacts entre deux neurones (entre axone et dendrite) se font par l'intermédiaire des synapses.

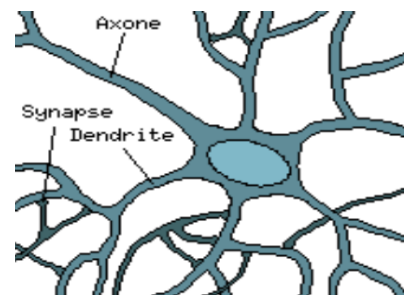


Figure 11: Neurone biologique

- Les signaux n'opèrent pas de manière linéaire : effet de seuil.

1.5. Neurone formel

En équivalence avec le système biologique, Ce processus est appelé neurone (neurone formel ou bien neurone artificiel).

1.5.1. Structure

Pour le neurone d'indice j :

Les entrées sur celui-ci sont de poids w_{ji}

Les connexions avals sont de poids w_{kj}

Le neurone reçoit les entrées $x_1, \dots, x_i, \dots, x_n$.

Le potentiel d'activation du neurone j est défini comme la somme pondérée (les poids sont les coefficients synaptiques w_i) des entrées $a_j = \sum(w_{ji} \cdot x_i)$.

La sortie obtenue après application d'une fonction de transfert est définie par $y_j = g(a_j)$, cette fonction de transfert calcule la valeur de l'état du neurone (sortie). C'est cette valeur qui sera transmise aux neurones avals.

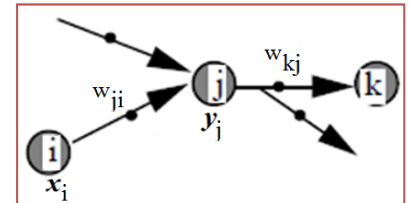
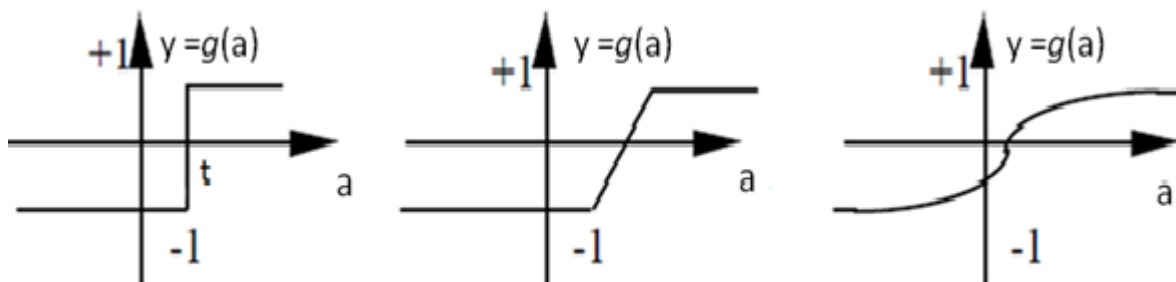


Figure 12: Neurone formel

1.5.2. La fonction de transfert

Il existe de nombreuses formes possibles :



Fonction à seuil

linéaire par morceaux

sigmoïde

Figure 13: Représentation des différentes fonctions de transfert

1.6. Le perceptron

Le perceptron de Rosenblatt (1958) est le premier RNA opérationnel. C'est un réseau à propagation avant avec seulement deux couches (entrée et sortie) entièrement interconnectées. Il est composé de neurones à seuil.

L'apprentissage est supervisé et les poids sont modifiés selon la règle delta.

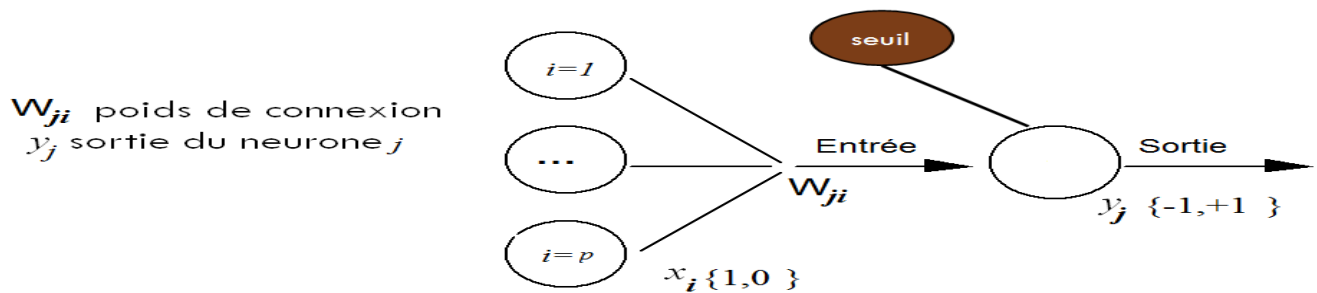


Figure 14:Exemples du perceptron

Un réseau de neurones monocouches est caractérisé par :

- p informations en entrée
- q neurones
- chacun des q neurones est connecté aux p informations d'entrée

A noter :

$X = (x_i) 1 \leq i \leq p$: les p informations d'entrée

$w_{ji}, 1 \leq i \leq p \text{ et } 1 \leq j \leq q$: les poids de connexion

y_j : la sortie du j-ème neurone

a_j : la donnée d'entrée (somme pondérée) du j-ème neurone.

seuil : seuil d'activation du neurone

On parle aussi du coefficient de biais w_{j0} , pour ajuster la sensibilité du neurone

On a donc l'équation suivante :

$$\forall 1 \leq j \leq q \quad y_j = g(a_j) = g\left(\sum_{i=0}^p (w_{ji}) * x_i\right)$$

Généralement, le neurone est activé si l'activation $a_j \geq 0$ on atteint le seuil de la fonction d'activation lorsque la somme pondérée des informations d'entrée vaut le coefficient de biais (w_{j0})

$$a_j = \sum_{i=1}^p (w_{ji}) * x_i - w_{j0} \geq 0$$

$$\sum_{i=1}^p (w_{ji}) * x_i \geq w_{j0}$$

1.6.1. Apprentissage

L'apprentissage est une phase du développement d'un réseau de neurones durant laquelle le comportement du réseau est modifié jusqu'à l'obtention du comportement désiré.

On distingue deux grandes classes d'algorithmes d'apprentissage :

- **L'apprentissage supervisé** : les coefficients synaptiques sont évalués en minimisant l'erreur (entre sortie souhaitée et sortie obtenue) sur une base d'apprentissage.
- **L'apprentissage non supervisé** : on ne dispose pas de base d'apprentissage. Les coefficients synaptiques sont déterminés par rapport à des critères de conformité : spécifications générales.

a. Apprentissage par l'algorithme du perceptron

L'algorithme du perceptron :

On note S la base d'apprentissage.

S est composée de couples (x, c) où :

x est le vecteur associé à l'entrée (x_0, x_1, \dots, x_n)

c la sortie correspondante souhaitée

On cherche à déterminer les coefficients (w_0, w_1, \dots, w_n).

- Initialiser aléatoirement les coefficients w_i .
- Répéter :
 - Prendre un exemple (x, c) dans S
 - Calculer la sortie o du réseau pour l'entrée x
 - Mettre à jour les poids :
 - Pour i de 0 à n :
 - $w_i = w_i + \epsilon * (c - o) * x_i$
 - Fin Pour
- Fin Répéter

b. Apprentissage par l'algorithme du perceptron : exemple

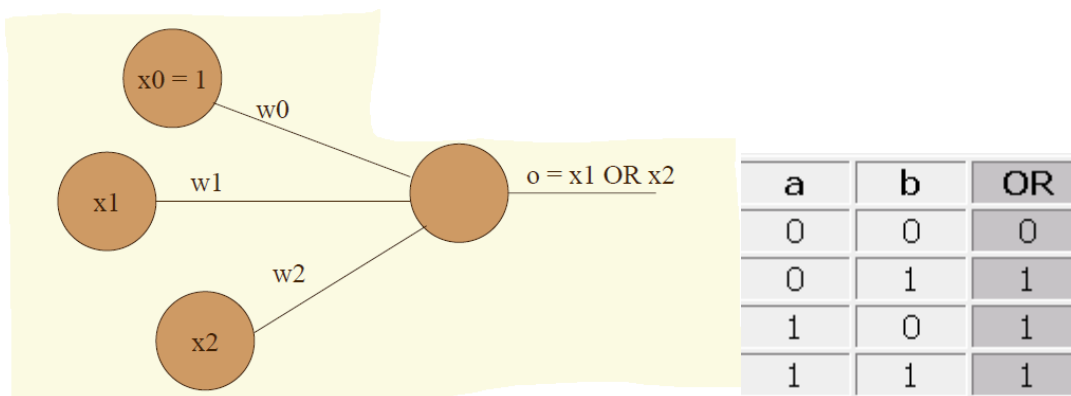


Figure 15: Exemple d'apprentissage

$\epsilon = 1$

x_0 vaut toujours 1

Initialisation : $w_0 = 0$; $w_1 = 1$; $w_2 = -1$

Étape	w_0	w_1	w_2	Entrée	$\sum_0^2 w_i x_i$	o	c	w_0	w_1	w_2
init								0	1	-1
1	0	1	-1	100	0	0	0	$0+0x1$	$1+0x0$	$-1+0x0$
2	0	1	-1	101	-1	0	1	$0+1x1$	$1+1x0$	$-1+1x1$
3	1	1	0	110	2	1	1	1	1	0
4	1	1	0	111	2	1	1	1	1	0
5	1	1	0	100	1	1	0	$1+(-1)x1$	$1+(-1)x0$	$0+(-1)x0$
6	0	1	0	101	0	0	1	$0+1x1$	$1+1x0$	$0+1x1$
7	1	1	1	110	2	1	1	1	1	1
8	1	1	1	111	3	1	1	1	1	1
9	1	1	1	100	1	1	0	$1+(-1)x1$	$1+(-1)x0$	$1+(-1)x0$
10	0	1	1	101	1	1	1	0	1	1

Figure 16: Exemple de résultat du calcul

Donc : $w_0 = 0$; $w_1 = 1$; $w_2 = 1$

Ce perceptron calcule le OU logique pour tout couple (x_1 ; x_2)

c. Règles d'apprentissage

L'apprentissage consiste à modifier le poids des connexions

entre les neurones.

Il existe plusieurs règles de modification :

- **Loi de Hebb** : $\Delta w_{ji} = \alpha x_i y_j$
- **Règle de delta**: $\Delta w_{ji} = \alpha (d_j - y_j) x_i$
- **Règle de Grossberg** : $\Delta w_{ji} = \alpha (x_i - w_{ji}) y_j$

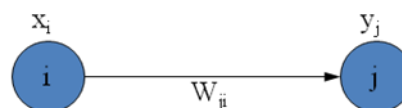


Figure 17: Présentation du poids de connexion

d. Apprentissage supervisé

Deux algorithmes pour "faire apprendre" à un réseau de neurones monocouche.

- Apprentissage par descente de gradient
- Apprentissage par l'algorithme de Windrow-Hoff

Ils consistent à comparer le résultat qui était attendu par les exemples puis à minimiser l'erreur commise sur les exemples.

1.6.2. Limitation du perceptron

En général, les perceptrons ne permettent de représenter que des fonctions linéairement séparables

Pour les fonctions non linéairement séparables, il faut établir un réseau de neurones multicouche.

1.7. Le Perceptron multicouche

C'est un réseau à couches cachées, le flux se fait "en avant". En pratique une ou deux couches

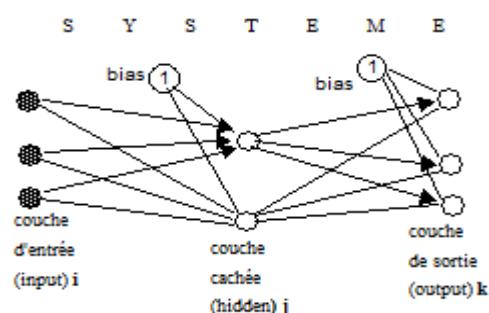


Figure 18: Exemple du perceptron multi couche

cachées suffisent

1.7.1. Apprentissage

Algorithme de rétro propagation

- Utilise la règle de modification des poids ("delta rule")
- Remonte couche par couche, des neurones de sortie vers les neurones d'entrées
- Modifie les poids synaptiques en amont de chaque couche, de manière à diminuer l'erreur commise en sortie
- La fonction d'activation doit être indéfiniment dérivable

La fonction sigmoïde $f(x) = \frac{1}{1 + e^{-x}}$

1.7.2. Algorithme de rétropropagation

L'algorithme est basé sur :

- Propagation vers l'avant des entrées $x(n)$
- Rétropropagation de l'erreur entre $d(n)$ et $y(n)$ vers l'arrière pour l'adaptation des poids

Modèle du neurone j :

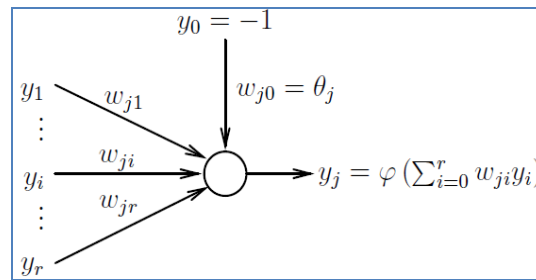


Figure 19:Modèle de neurone j

$\phi [.]$ fonction d'activation

L'erreur pour le neurone j : $e_j(n) = d_j(n) - y_j(n)$

La somme des erreurs quadratiques $E(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n)$

Pour diminuer les erreurs on modifie les poids $w_{ji}(n)$

Descente du gradient de l'erreur $\frac{\partial E(n)}{\partial w_{ji}(n)}$

La variation de poids $\Delta w_{ji} = -\eta \frac{\partial E(n)}{\partial w_{ji}(n)}$

$0 \leq \eta \leq 1$ taux d'apprentissage ou gain de l'algorithme

1.7.3. Sommaire de l'algorithme

1. Initialiser tous les poids à des valeurs aléatoires dans l'intervalle $[-0.5, 0.5]$;
2. Normaliser les données d'entraînement ;
3. Permuter aléatoirement les données d'entraînement ;
4. Pour chaque donnée d'entraînement n :
 - a. Calculer les sorties observées en propageant les entrées vers l'avant ;
 - b. Ajuster les poids en rétropropageant l'erreur observée :

$$w_{ji}(n) = w_{ji}(n-1) + \Delta w_{ji}(n) = w_{ji}(n-1) + \eta \delta_j(n) y_i(n)$$

Où le "gradient local" est défini par :

$$\delta_j(n) = \begin{cases} e_j(n) y_j(n) [1 - y_j(n)] \\ y_j(n) [1 - y_j(n)] \sum_k \delta_k(n) w_{kj}(n) \end{cases}$$

→ Si j ∈ couche sortie

→ Si j ∈ couche cachée

Répéter les étapes 3 et 4 jusqu'à un nombre maximum d'itérations ou jusqu'à ce que la racine de l'erreur quadratique moyenne (EQM) soit inférieure à un certain seuil.

1.7.4. Applications : PMC

Classification :

Reconnaissance de formes :

- les lettres de l'alphabet
- les chiffres de 0 à 9

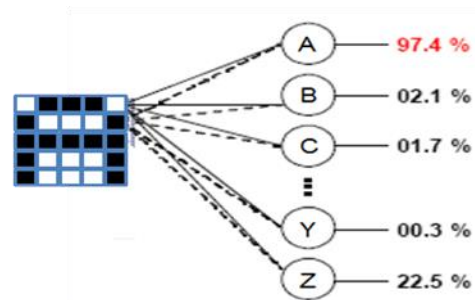


Figure 20: Réseau de neurone et reconnaissance de forme

2. Arbre de décision

2.1. Définition

L'arbre de décision est un arbre au sens informatique du terme. Il est représenté sous la forme graphique d'un arbre, d'une arborescence, ou encore d'un diagramme illustrant des règles de décision. C'est un outil d'aide à la décision, qui représente graphiquement un séquençage logique, faisant apparaître les différents résultats possibles, en fonction des choix effectués à chaque étape.

Sa lisibilité, sa rapidité d'exécution et le peu d'hypothèses nécessaires a priori expliquent sa popularité actuelle.

Il existe deux catégories d'arbre de décision en Data Mining:

- **Les arbres de classification (Clustering Tree)** permettent de prédire à quelle classe la variable-cible appartient, dans ce cas la prédiction est une étiquette de classe,
- **Les arbres de régression (Regression Tree)** permettent de prédire une quantité réelle (par exemple, le prix d'une maison ou la durée de séjour d'un patient dans un hôpital), dans ce cas la prédiction est une valeur numérique [20].

2.2. Constitution

Un arbre de décision est un classifieur représenté sous forme d'arbre tel que :

- **Les nœuds de l'arbre** qui permettent de tester les attributs.
- Il est composé de **branches** qui représentent chacune une valeur de l'attribut testé.

- Il peut également être constitué de feuilles, c'est-à-dire **les nœuds terminaux**, qui indiquent la classe résultante.

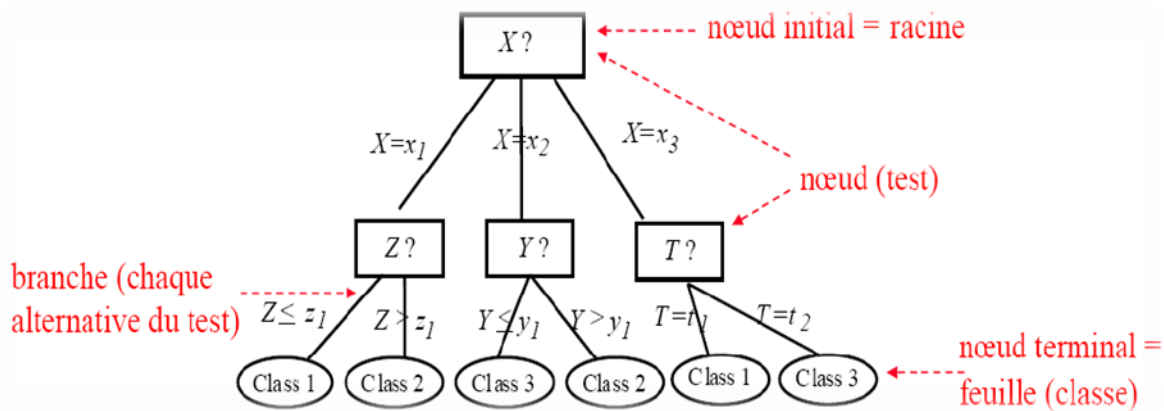


Figure 21: Représentation d'un arbre de décision

Le chemin de la racine jusqu'à la feuille est appelé "règle de décision" [20].

2.3. But d'arbre de décision

L'arbre de décision est mis en place dans un souci de prise de décision la plus efficiente.

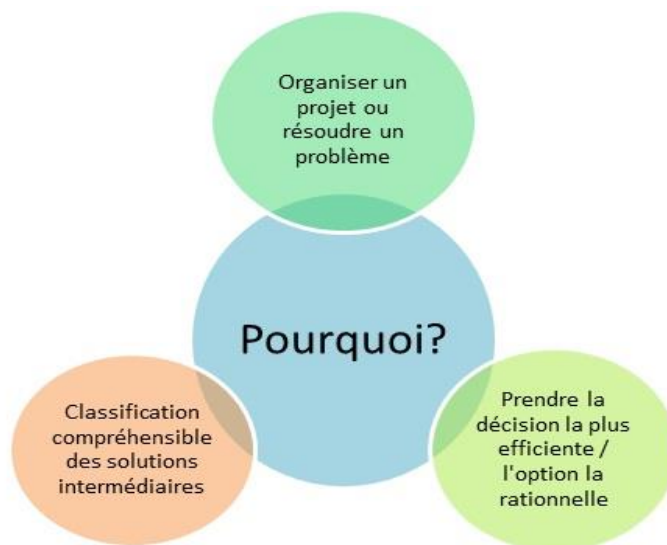


Figure 22: Représente le but d'arbre de décision

L'arbre de décision a pour but de mettre sous forme lisible une succession de choix afin de pouvoir choisir la solution la plus appropriée. Cette méthodologie peut alors être qualifiée d'outil d'aide à la décision [20].

2.4. Les domaines d'application

L'arbre de décision est principalement utilisés dans des domaines divers et variés tels que : datamining, accord de crédit, imagerie, diagnostic médical, marketing ciblé, sécurité et détection des fraudes fiscales [20].

2.5. La construction d'un arbre de décision

Le but de la construction d'un arbre de décision est de déterminer les meilleurs attributs à placer à chaque nœud, afin d'obtenir un arbre le plus petit possible et qu'il permette d'en retirer les meilleures prédictions.

Reprenons cet exemple, son objectif est d'expliquer le comportement de joueur de Tennis à partir de prévisions météorologiques [21].

Nous possédons les critères suivants :

Tableau 3: Les critères de la base de données

Paramètres	Réponses possibles
Temps	Ensoleillé, Couvert, Pluvieux
Température	Chaude, Modérée, Fraiche
Humidité	Elevée, Normale
Vent	VRAI, FAUX

Ainsi que nos bases de données (14 exemples sont présents).

Attributs prédictifs					Attribut de classes
N°:	Temps	Température	Humidité	Vent	Jouer
J1	Ensoleillé	Chaude	Elevée	FAUX	Non
J2	Ensoleillé	Chaude	Elevée	VRAI	Non
J3	Couvert	Chaude	Elevée	FAUX	Oui
J4	Pluvieux	Modérée	Elevée	FAUX	Oui
J5	Pluvieux	Fraiche	Normale	FAUX	Oui
J6	Pluvieux	Fraiche	Normale	VRAI	Non
J7	Couvert	Fraiche	Normale	VRAI	Oui
J8	Ensoleillé	Modérée	Elevée	FAUX	Non
J9	Ensoleillé	Fraiche	Normale	FAUX	Oui
J10	Pluvieux	Modérée	Normale	FAUX	Oui
J11	Ensoleillé	Modérée	Normale	VRAI	Oui
J12	Couvert	Modérée	Elevée	VRAI	Oui
J13	Couvert	Chaude	Normale	FAUX	Oui
J14	Pluvieux	Modérée	Elevée	VRAI	Non

Figure 23: Le comportement de joueur de Tennis à partir de prévisions météorologiques

Nous avons donc un ensemble de 14 exemples, afin de désigner le meilleur attribut nous devons calculer :

- **L'entropie :**

$$\text{Entropie}(S) = -p/N \log_2 p/N - n/N \log_2 n/N$$

- ❖ Où S est l'ensemble des exemples, de taille N,
- ❖ p (exemple positive) est le nombre d'exemples classés Oui,
- ❖ et n (exemple négative) est le nombre d'exemples classés Non dans l'ensemble S des N exemples.

Dans nos exemples :

Nœud contenant 9 exemples + et 5 exemples - :

$$\text{Entropie}(S) = -p/N \log_2 p/N - n/N \log_2 n/N$$

$$\text{Entropie}(S) = - (9/14) \log_2 (9/14) - (5/14) \log_2 (5/14)$$

$$\text{Entropie}(S) = 0.940$$

Si l'entropie vaut 0, alors tous les exemples appartiennent à la même classe (par exemple Oui). Si l'entropie vaut 1, alors c'est qu'il y a autant d'exemples positifs que d'exemples négatifs.

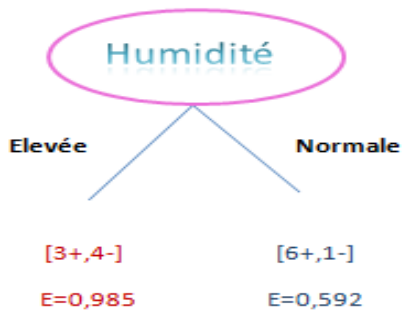
- **Gain d'information pour chacun de ces attributs :**

$$\text{Gain}(S, A) = \text{Entropie}(S) - \text{Somme sur les valeurs de A de } (|S_v| * \text{Entropie}(S_v) / |S|)$$

$$\text{Gain}(S, A) = E(S) - \sum \left(\frac{|S_v|}{|S|} \times E(S_v) \right)$$

- ❖ Où S = l'ensemble des exemples ,
- ❖ A est l'attribut utilisé,
- ❖ S_v = le sous-ensemble de S dont l'attribut A à la valeur v.

Considérons l'attribut "Humidité" :

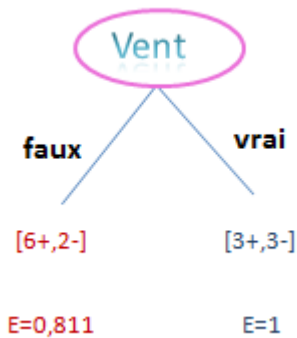


$$\text{Gain}(S, \text{Humidité}) = 0,940 - (7/14)0,985 - (7/14)0,592$$

$$\text{Gain}(S, \text{Humidité}) = 0,151$$

Figure 24: L'attribut humidité

Les mêmes calculs pour l'attribut "Vent" :

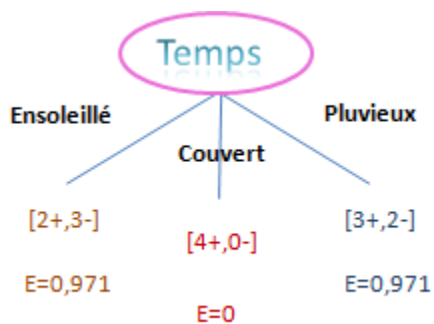


$$\text{Gain}(S, \text{Vent}) = 0,940 - (8/14)0,811 - (6/14)1$$

$$\text{Gain}(S, \text{Vent}) = 0,048$$

Figure 25: L'attribut vent

Pour l'attribut "Temps" :

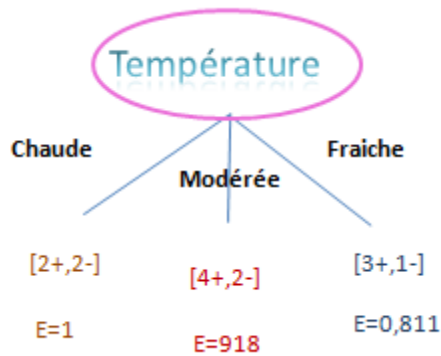


$$\text{Gain}(S, \text{Temps}) = 0,940 - (5/14)0,971 - (5/14)0,971 - 0$$

$$\text{Gain}(S, \text{Temps}) = 0,246$$

Figure 26: L'attribut temps

Pour l'attribut "Température" :



$$\text{Gain}(S, \text{Température}) = 0,940 - (4/14)1 - (6/14)0,918 - (4/14)0,811$$

$$\text{Gain}(S, \text{Température}) = 0,029$$

Figure 27: L'attribut température

Alors voici la valeur du gain pour chaque attribut, on va choisir l'attribut permettant le gain d'information le plus important donc ici l'attribut gagnant est l'attribut **temps**.

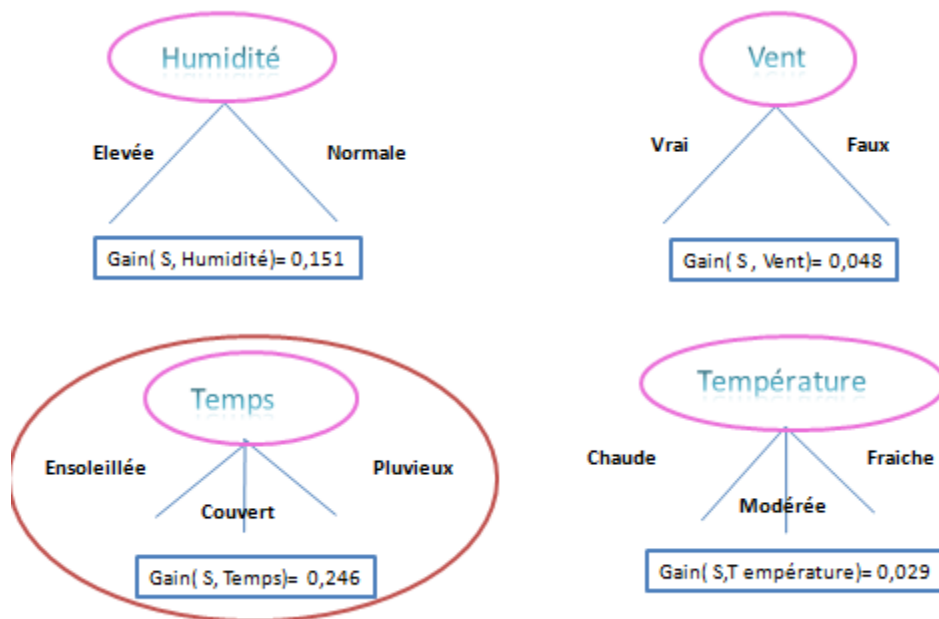
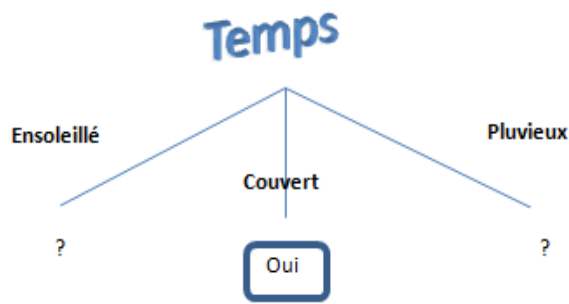


Figure 28: La valeur du gain pour chaque attribut

Après avoir choisi le premier attribut, la question qui se pose ici c'est quel attribut sera produit par la modalité ensoleillé de la variable temps.

La même chose on va calculer le gain pour chaque attribut et on choisit celui qui a la valeur la plus importante.



$$\text{Gain}(\text{SSoleil}, \text{Humidité}) = 0,970 - (3/5) 0 - (2/5) 0$$

$$\text{Gain}(\text{SSoleil}, \text{Humidité}) = 0,970$$

$$\text{Gain}(\text{SSoleil}, \text{Température}) = 0,970 - (2/5) 0 - (2/5) 1 - (1/5) 0$$

$$\text{Gain}(\text{SSoleil}, \text{Température}) = 0,570$$

$$\text{Gain}(\text{SSoleil}, \text{Vent}) = 0,970 - (2/5) 1 - (3/5) 0,918$$

$$\text{Gain}(\text{SSoleil}, \text{Vent}) = 0,019$$

Figure 29: Construction des sommets

La variable gagnante est humidité

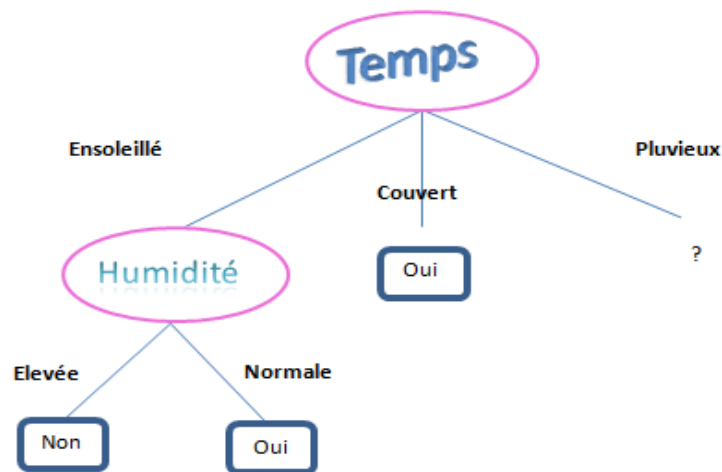
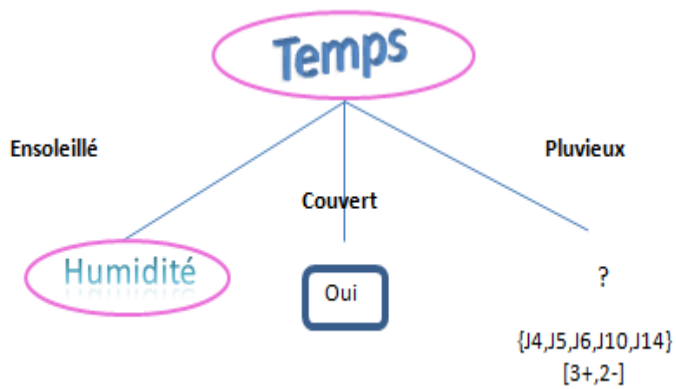


Figure 30: Construction du premier sommet

Donc maintenant qu'on a construit le premier sommet, il reste à voir l'attribut qui sera produit par la modalité pluvieux de la variable temps.

La même chose on va calculer le gain pour chaque attribut et on va choisir celui qui a la valeur la plus importante.



$$\text{Gain (SPluie,Humidité)} = 0,970 - (2/5) 1 - (3/5) 0,918$$

$$\text{Gain(SPluie,Humidité)} = 0,019$$

$$\text{Gain(SPluie,Température)} = 0,970 - (0/5) - (3/5) 0,918 - (2/5) 1$$

$$\text{Gain(SPluie,Température)} = 0,019$$

$$\text{Gain(SPluie,Vent)} = 0,970 - (2/5) 0 - (3/5) 0$$

$$\text{Gain(SPluie,Vent)} = 0,970$$

Figure 31:Construction du deuxième sommet

La variable gagnante est vent.

Enfin voici l'arbre obtenu,

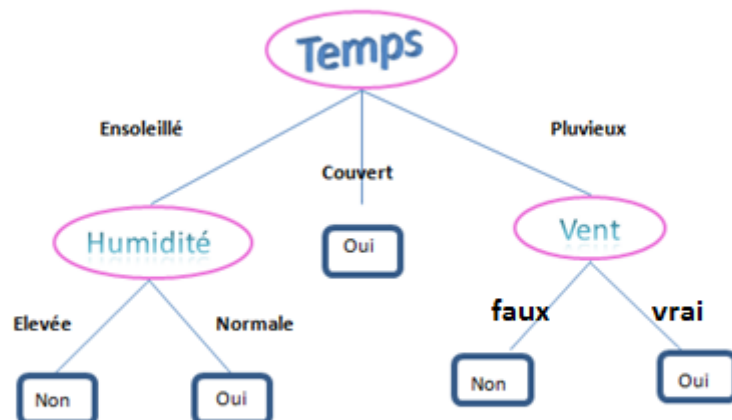


Figure 32:L'arbre obtenu

Enfin voici l'arbre obtenu par logiciel weka, on remarque que c'est le même arbre que nous avons construit.

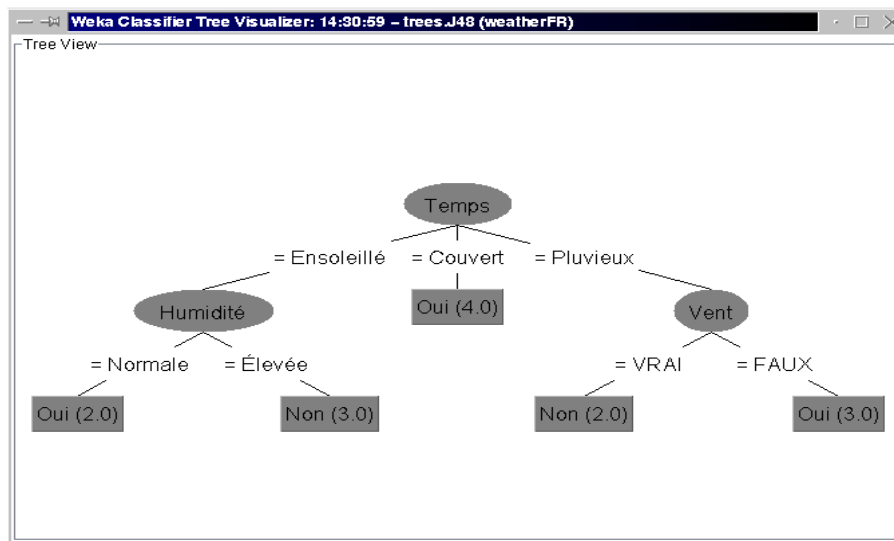


Figure 33:L'arbre obtenu par logiciel weka

2.6. Les avantages

Comparativement à d'autres méthodes de Data Mining, les arbres de décision présentent plusieurs avantages:

- La simplicité de compréhension et d'interprétation. Ils ont l'avantage d'être plus faciles à comprendre que les réseaux de neurones qui gardent le côté « boîte noire »,
- Faible de préparation des données (pas de normalisation, de valeurs vides à supprimer, ou de variable muette),
- Le modèle peut gérer à la fois des valeurs numériques et des catégories. D'autres techniques sont souvent spécialisées sur un certain type de variables (les réseaux neuronaux ne sont utilisables que sur des variables numériques),
- Pas de paramétrage difficile pour la construction de l'arbre,
- Rapidité et capacité à traiter des très grandes bases.

2.7. Les inconvénients

En revanche, les arbres de décision présentent certains inconvénients :

- Problème de stabilité sur les petites bases de données (feuilles à très petits effectifs),
- Certains concepts sont difficiles à exprimer à l'aide d'arbres de décision (comme XOR). Dans ces cas, les arbres de décision deviennent extrêmement larges,
- Non incrémental: recommencer la construction de l'arbre si on veut intégrer de nouvelles données,
- Sensibilité au bruit et points aberrants.

2.8. Exemple des méthodes qui aide à la construction d'un arbre

Parmi les méthodes qui peuvent nous aider pour la construction de notre arbre de décision nous citons [20]:

- ID3 (Inductive Decision Tree)
- C4.5, C5 (successeurs d'ID3)
- CART (Classification and Regression Tree)
- QUEST (Quick Unbiased Efficient Statistical Trees)
- CHAID (Chi-Square Automatic Interaction Detection)

ID3 et CART ont été inventées de manière indépendante dans les décennies 1970-1980, mais utilisent des approches similaires pour apprendre des arbres de décision depuis l'ensemble d'apprentissage.

Tous ces algorithmes se distinguent par le ou les critères de segmentation utilisés, par les méthodes d'élagages implémentées, par leur manière de gérer les données manquantes dans les prédicateurs.

3. SVM

3.1. Définition

Les machines à vecteurs de support sont un ensemble de techniques d'apprentissage destinées à résoudre des problèmes de discrimination, c'est à dire décider à quelle classe appartient un échantillon, ou de régression, c'est à dire prédire la valeur numérique d'une variable.

3.2. Principe de fonctionnement général

Un SVM, comme un perceptron, trouve un séparateur linéaire entre les points de données de deux classes différentes. En général, il peut y avoir plusieurs séparateurs possibles entre les classes (en supposant le problème linéairement séparable) et qu'un perceptron n'a pas de préférence parmi celles-ci. Dans les SVMs, cependant, nous faisons un choix particulier parmi tous les séparateurs possibles : nous voulons celui avec la "marge" maximale.

3.3. L'histoire

Les séparateurs à vastes marges reposent sur deux idées clés : la notion de marge maximale et la notion de fonction noyau. Ces deux notions existaient depuis plusieurs années avant qu'elles ne soient mises en commun pour construire les SVM.

L'idée des hyperplans à marge maximale a été explorée dès 1963 par Vladimir Vapnik et A. Lerner, et en 1973 par Richard Duda et Peter Hart dans leur livre *Pattern Classification*. Les fondations théoriques des SVM ont été explorées par Vapnik et ses collègues dans les années 70 avec le développement de la théorie de Vapnik-Chervonenkis, et par Valiant et la théorie de l'apprentissage PAC.

L'idée des fonctions noyaux n'est pas non plus nouvelle : le théorème de Mercer date de 1909, et l'utilité des fonctions noyaux dans le contexte de l'apprentissage artificiel a été montrée dès 1964 par Aizermann, Braverman et Rozoener.

Ce n'est toutefois qu'en 1992 que ces idées seront bien comprises et rassemblées par Boser, Guyon et Vapnik dans un article, qui est l'article fondateur des séparateurs à vaste marge. L'idée des variables ressorts, qui permet de résoudre certaines limitations pratiques importantes, ne sera introduite qu'en 1995. À partir de cette date, qui correspond à la publication du livre de Vapnik, les SVM gagnent en popularité et sont utilisés dans de nombreuses applications.

Un brevet américain sur les SVM est déposé en 1997 par les inventeurs originaux.

3.4. Notions de base

a. Hyperplan

Plaçons-nous dans le cas d'une classification binaire (i.e. les exemples à classer réparties en 2 classes). On appelle hyperplan séparateur un hyperplan qui sépare les deux classes figure 34, en particulier il sépare leurs points d'apprentissage. Comme il n'est en générale pas possible d'en

trouver un, on se contentera donc de chercher un hyperplan discriminant qui est une approximation au sens d'un critère à fixer (maximiser la distance entre ces deux classes).

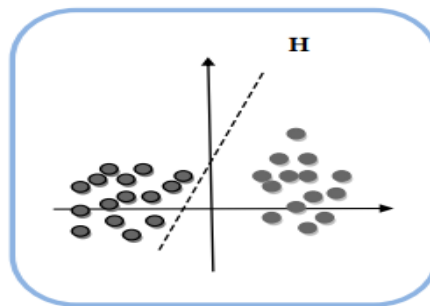


Figure 34: L'hyperplan H qui sépare les deux ensembles de points.

b. Vecteurs de support

Pour une tâche de détermination de l'hyperplan séparable des SVM est d'utiliser seulement les points les plus proches (i.e. les points de la frontière entre les deux classes des données) parmi l'ensemble total d'apprentissage, ces points sont appelés vecteurs de support.

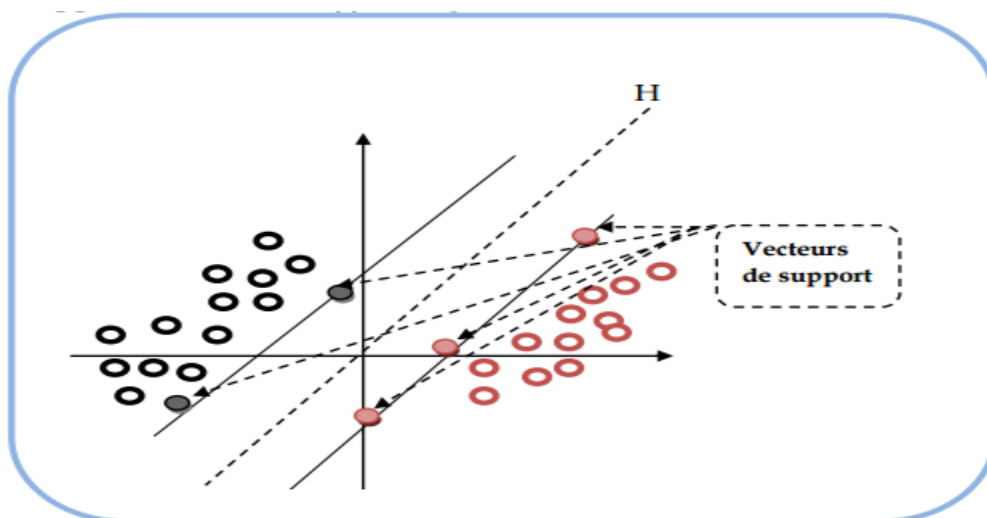


Figure 35: Les vecteurs de support.

c. Marge

Il existe une infinité d'hyperplans capable de séparer parfaitement les deux classes d'exemples. Le principe des SVM est de choisir celui qui va maximiser la distance minimale entre l'hyperplan et les exemples d'apprentissage (i.e. la distance entre l'hyperplan et les vecteurs de support), cette distance est appelée la marge.

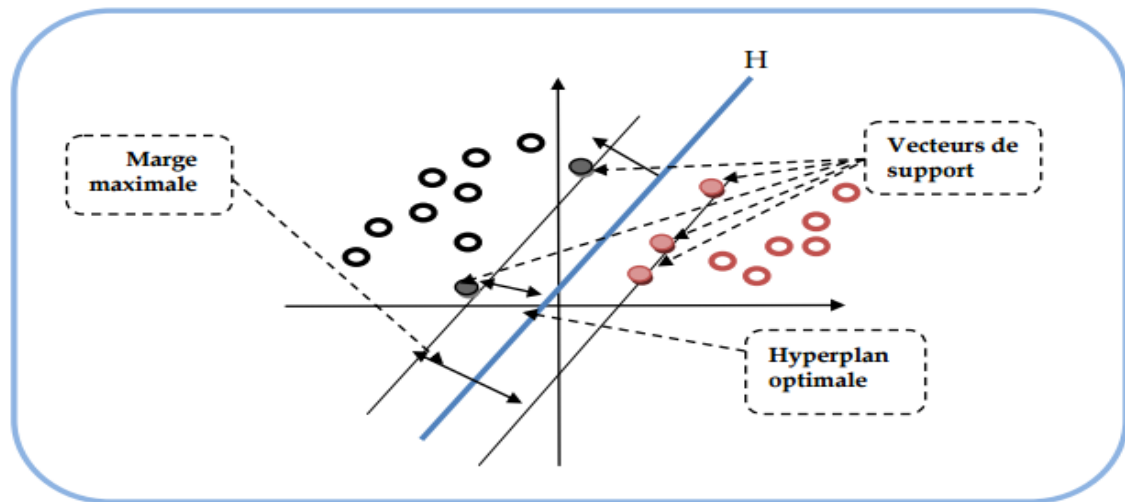


Figure 36: Hyperplan optimal, vecteurs de support et marge maximale.

3.5. Les domaines d'applications

SVM est une méthode de classification qui montre de bonnes performances dans la résolution des problèmes variés. Cette méthode a montré son efficacité dans de nombreux domaines d'applications tels que: bioinformatique, recherche d'information, vision par ordinateur, finance.

Les SVMs ont pour rôle de :

- Classification de données biologiques/physiques,
- Classification de documents numériques,
- Classification d'expressions faciales,
- Classification de textures E-learning,
- Détection d'intrusion,
- Reconnaissance de la parole,

3.6. Les avantages

SVM est une méthode de classification intéressante car le champ de ses applications est large, parmi ses avantages nous avons :

- Un grand taux de classification et de généralisation par rapport aux méthodes classiques.
- Elle nécessite moins d'effort pour designer l'architecture adéquate (petit nombre de paramètres à régler ou à estimer).

- La résolution du problème est convertie en résolution d'un problème quadratique convexe dont la solution est unique et donnée par des méthodes mathématiques classiques de programmation quadratique.

3.7. Inconvénients

L'inconvénient majeur du classificateur SVM est qu'il est désigné ou conçu pour la classification binaire (la séparation entre deux classes une +1 et l'autre -1) [22].

4. Les Réseaux bayésiens

4.1 Définition

Un Réseau bayésiens est un graphe orienté, acyclique, dont les nœuds sont des variables aléatoires et les arcs représentent [23]:

- des dépendances.
- des distributions de probabilités conditionnelles.

Il y a d'autres appellations pour les réseaux bayésiens:

- ❖ Réseaux de croyance (belief networks),
- ❖ Model graphique dirigé.

4.2. Classification naïve bayésienne

C'est un type de classification Bayésienne probabiliste simple basée sur le théorème de Bayes avec une forte indépendance (dite naïve) des hypothèses [14].

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$

- Ou H est l'hypothèse à tester et E l'évidence associée à H, P(H) est une probabilité a priori

4.3. Exemples

Voilà notre base de données de test :

Outlook {sunny, overcast, rainy}

Temperature numeric

Humidity numeric

Windy {TRUE, FALSE}

Play {yes, no}

@data

Sunny, 85, 85, FALSE, no

sunny, 80, 90, TRUE, no

overcast, 83, 86, FALSE, yes

rainy, 70, 96, FALSE, yes

rainy, 68, 80, FALSE, yes

rainy, 65, 70, TRUE, no

overcast, 64, 65, TRUE, yes

sunny, 72, 95, FALSE, no

sunny, 69, 70, FALSE, yes

rainy, 75, 80, FALSE, yes

sunny, 75, 70, TRUE, yes

overcast, 72, 90, TRUE, yes

overcast, 81, 75, FALSE, yes

rainy, 71, 91, TRUE, no

➤ Calcul des probabilités conditionnelles à partir des instances

- $P(E|H) \rightarrow P(\text{outlook : sunny}|\text{yes}), P(\text{windy : TRUE}|\text{yes}), \dots$
- $P(H) \rightarrow P(\text{yes}), P(\text{no})$

Résultat :

- $P(\text{outlook : sunny}|\text{yes}) = 2/9$
- $P(\text{outlook : sunny}|\text{no}) = 3/5$
- $P(\text{windy : TRUE}|\text{yes}) = 3/9$
- ...
- $P(\text{yes}) = 9/14$
- $P(\text{no}) = 5/14$

Tableau 4: Résultat obtenu à l'aide de l'algorithme naïve bayes.

Outlook			Temperature			Humidity			windy			play	
Value	Yes	no	Value	yes	no	Value	Yes	No	Value	Yes	No	Yes	no
Sunny	2/9	3/5	Hot	2/9	2/5	High	3/9	4/5	False	6/9	2/5	9/14	5/14
Overcast	4/9	0/5	Mild	4/9	2/5	normal	6/9	1/5	true	3/9	3/5		
rainy	3/9	2/5	cool	3/9	1/5								

Un nouvel exemple arrive : (sunny ; cool ; high ; true ;?)

C'est-à-dire → outlook: sunny; temperature: cool; humidity: high; windy: true

$$P(\text{yes} | E) = \frac{P(\text{sunny}|\text{yes}) * P(\text{cool}|\text{yes}) * P(\text{high}|\text{yes}) * P(\text{true}|\text{yes}) * P(\text{yes})}{P(E)}$$

$$P(\text{yes} | E) = \frac{\frac{2}{9} * \frac{3}{9} * \frac{3}{9} * \frac{3}{9} * \frac{9}{14}}{P(E)}$$

Autrement :

- Vraisemblance de yes : $\frac{2}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{9}{14} = 0.0053$
- Vraisemblance de no : $\frac{3}{5} \times \frac{1}{5} \times \frac{4}{5} \times \frac{3}{5} \times \frac{5}{14} = 0.0206$

Les nombres peuvent être changés en probabilités

- Probabilité de yes = $0.0053 / (0.0053 + 0.0206) = 20.5\%$
- Probabilité de no = $0.0206 / (0.0053 + 0.0206) = 79.5\%$

Tableau 5: Résultat de calcul

Outlook			Temperature			Humidity			windy			play	
Value	Yes	no	Value	yes	no	Value	Yes	No	Value	Yes	No	Yes	no
Sunny	2/9	3/5	Hot	2/9	2/5	High	3/9	4/5	False	6/9	2/5	9/14	5/14
Overcast	4/9	0/5	Mild	4/9	2/5	normal	6/9	1/5	true	3/9	3/5		
rainy	3/9	2/5	cool	3/9	1/5								

V. Conclusion

Les réseaux de neurones, les arbres de décision, réseaux bayésiens...etc. sont utilisés en analyse décisionnelle dans sa partie datamining pour la prédiction, la classification et l'analyse des données. Ces techniques ont marqué leur grande importance dans l'informatique décisionnelle du fait qu'aujourd'hui, notamment dans les grandes administrations et les groupes internationaux, toutes les fonctions de l'entreprise utilisent des moyens d'aide à la décision pour assurer la pérennité de l'activité.

Chapitre5 : Réalisation

I. Introduction

Pour développer notre application de prédiction de bug pour les logiciels orientés objet, nous allons passer par le choix de la base de données appropriée, puis l'application de quelques méthodes du DataMining pour la prédiction, et enfin l'étude de convergence de ces méthodes. Pour cet objectif nous allons commencer par citer les différents outils de développement utilisés, puis décrire toutes les bases de données proposées et plus particulièrement celle utilisée, enfin une vue global sur l'application.

II. Base de données

1. Quelques bases de données du génie logiciel

Nous présentons ci dessous l'ensemble des bases de données, trouvées, sous forme d'un tableau contenant leurs noms, leurs sites officiels et leurs descriptions:

Tableau 6:Description des bases de données du génie logiciel

Nom	Site	Open	Format	description
FLOSS Metrics	http://flossmetrics.org/	oui	DB dumps, Web srvs, Web	permet la collecte des données relatives au code source, aux mails et aux rapports de bogue. il est possible que les métriques supportées par la base de données ne soient pas suffisantes pour l'étude de l'évolution des logiciels.
PROMIS (PRedictOr Models In Software Engineering)	http://promisedata.org/	Oui	Mainly ARFF, CSV, others	Son objectif est d'être un centre de stockage à long terme pour les données SE (software Engineering).

Qualitas Corpus (QC)	http://qualitascorp.com/	oui	Code, JAR, CSV	Le Corpus Qualitas est une sélection pointue de systèmes logiciels destinés à être utilisés pour les études empiriques des artefacts de code. L'objectif principal est de fournir une ressource qui prend en charge les études reproductibles de logiciel.
Sourcerer Project	http://sourcerer.ics.uci.edu/	Oui	Java Source code + DB structure	un projet de recherche à l'Université de Californie, Irvine qui visent à explorer le phénomène open source grâce à l'utilisation de l'analyse du code.
Ultimate Debian Database (UDD)	http://udd.debian.org/	Oui	DB dumps	Base de données ultime Debian (UDD) rassemble un grand nombre de données sur les divers aspects de Debian dans la même base de données SQL. Il permet aux utilisateurs d'accéder facilement et de combiner toutes ces données.
Bug Prediction Dataset (BPD)	http://bug.inf.usi.ch/	Oui	CSV	L'ensemble de données de prédiction de bug est une collection de modèles et de mesures des systèmes de logiciels et de leurs histoires. Le but d'un tel ensemble de

				données est de permettre aux gens de comparer différentes approches de prédiction de bug et d'évaluer si une nouvelle technique est une amélioration par rapport existants.
International Software Benchmarking Standards Group (ISBSG):	http://www.isbsg.org/	Non	MS Excel Spreadsheet	Permet d'améliorer la gestion des ressources informatiques à la fois par les entreprises et le gouvernement à travers la prévision et l'exploitation des dépôts publics de connaissances en génie logiciel qui sont normalisés, vérifiés, récents et représentatifs des technologies actuelles. Son rôle : - Recueil des données - Analyse des données - Résultats de la recherche de paquets - Le partage des connaissances grâce à des produits et services.
Eclipse Bug Data (EBD)	http://www.st.cs.uni-saarland.de/software/bug-data/eclipse/	Oui	ARFF and CSV (same info)	L'utilisation typique de ces données est de valider les hypothèses sur la nature et la cause des erreurs comme ils se produisent au cours du développement de logiciel.

Software-artifact Infrastructure Repository (SIR)	http://sir.unl.edu/	Oui	Code for analysis and testing tech	SIR est un dépôt d'artefacts relatifs aux logiciels destinés à soutenir l'expérimentation contrôlée rigoureuse avec l'analyse des programmes et des techniques de tests de logiciels, et de l'éducation dans l'expérimentation contrôlée.
SourceForge Research Data Archive (SRDA)	http://zerlot.cse.nd.edu/	Oui	DB Dumps	Ce wiki contient des informations sur l'utilisation de l'archive de données de recherche SourceForge (SRDA).
Helix Data Set	http://www.ict.swin.edu.au/research/projects/helix/	Oui	CSV	Helix est un système de gestion de base de données pionnier pour la plateforme Apple Macintosh, créé en 1983. Helix utilise un « langage de programmation » graphique pour ajouter une logique à ses applications, permettant aux non-développeurs de construire des applications sophistiquées.
Tukutuku	http://www.metriq.biz/tukutuku/	Non	Effort prediction for Web apps	Le but du projet d'analyse comparative Tukutuku est double: premièrement, pour recueillir des données sur les projets Web à utiliser pour construire des modèles d'estimation des coûts génériques spécifiques à l'entreprise ou qui aidera une

				<p>entreprise Web améliorer ses pratiques d'estimation des coûts actuels; secondes, pour permettre à une entreprise de Web pour comparer sa productivité au sein et entre les entreprises Web.</p>
--	--	--	--	--

2. Base de données utilisée

2.1. Définition

Afin de bien atteindre notre objectif, nous avons choisi la base de données PROMISE (PRedictOr Models In Software Engineering), qui est un référentiel dans le génie logiciel. Où nous trouverons une collection d'ensembles de données et des outils accessibles au public à servir les chercheurs dans la construction de modèles prédictifs de logiciels et la communauté de génie logiciel au sens large [24].

2.2. Les raisons de choix de PROMISE

Les raisons derrière le choix de cette base de données sont [24] :

- Le référentiel est créé pour encourager des modèles prédictifs reproductibles, vérifiables, réfutables, et / ou améliorables du génie logiciel. Ces modèles pourraient être ciblés sur la planification, la conception, la mise en œuvre, les essais, l'entretien, l'assurance qualité, l'évaluation, l'amélioration des processus, la gestion, la prise de décision, et l'évaluation des risques dans les logiciels et le développement de systèmes.
- PROMISE se distingue de forums similaires avec son référentiel de données publiques et de se concentrer sur les détails méthodologiques, offrant un lieu interdisciplinaire unique pour l'ingénierie de logiciels et de communautés d'apprentissage de la machine, et la recherche pour les modèles de prédiction vérifiables et reproductibles qui sont utiles dans la pratique.
- Il donnera la priorité aux études empiriques basées sur des ensembles de données accessibles au public.

- Promise utilise 19 métriques logiciel, 6 métriques sont des **métriques de CK**. Ces métriques sont détaillées dans la section suivante.

2.3. Les métriques de PROMISE

Les métriques utilisées par PROMISE sont [25]:

- ❖ **WMC** (Weighted methods per class) : le nombre des méthodes par classe,
- ❖ **DIT** (Depth of Inheritance Tree): la profondeur d'héritage,
- ❖ **NOC** (Number of Children) : le nombre d'enfant d'une classe,
- ❖ **CBO** (Coupling between object classes): le nombre de classes couplées à une classe donnée
- ❖ **RFC** (Response for a Class) : le nombre de différentes méthodes qui peuvent être exécutées quand un objet de cette classe reçoit un message (lorsqu'une méthode est invoquée pour cet objet).
- ❖ **LCOM** (Lack of cohesion in methods): compte les ensembles de méthodes dans une classe qui ne sont pas liés par le partage d'une partie des champs de la classe.
- ❖ **Ca** (Afferent coupling) : C'est une mesure de la façon dont de nombreuses autres classes utilisent la classe spécifique. Couplage a la même définition dans le contexte de Ca que celui utilisé pour le calcul de CBO.
- ❖ **Ce** (Efferent coupling) : est une mesure de combien d'autres classes est utilisé par la classe spécifique. Couplage a la même définition dans le contexte de Ce que celui utilisé pour le calcul de CBO.
- ❖ **NPM** (Number of Public Methods for a class): le nombre des méthodes public d'une classe
- ❖ **LCOM3** (Lack of cohesion in methods Henderson-Sellers version):

$$LCOM3 = \frac{\left(\frac{1}{a} \sum_{j=1}^a \mu(A_j) \right) - m}{1 - m}$$

Elle est définie par cette relation :

LCOM3 varie entre 0 et 2.

Avec m : nombre de procédures (méthodes) en classe

a : nombre de variables (attributs en classe)

$\mu(A)$: nombre de méthodes qui accèdent à une variable (attribut)

- ❖ **LOC** (Lines of Code) : compte les lignes de code
- ❖ **DAM** (Data Access Metric) : est le rapport du nombre d'attribue privé (protégé) au nombre total d'attributs déclarés dans la classe.
- ❖ **MOA** (Measure of Aggregation) : est le nombre de déclarations de données (champs de classe), dont les types sont des classes définies par l'utilisateur.
- ❖ **MFA** (Measure of Functional Abstraction) : est le rapport du nombre de méthodes héritées par une classe et le nombre total de méthodes accessibles par des méthodes de membres de la classe.
- ❖ **CAMC** (Cohesion Among Methods of Class) : calcule le degré de parenté entre les méthodes d'une classe sur la base de la liste des paramètres des méthodes. La métrique est calculée en utilisant la somme des nombres de types différents de paramètres dans chaque méthode divisée par une multiplication du nombre de types différents de paramètres des méthodes de toute la classe et le nombre de méthodes.
- ❖ **IC** (Inheritance Coupling) : fournit le nombre de classes parentes à laquelle une classe donnée est couplée. Une classe est couplée à sa classe parent si l'une des conditions suivantes est satisfaite:
 - Une de ses méthodes héritées utilise un (ou un membre de données) variable qui est défini dans une nouvelle méthode / redéfini.
 - Une de ses méthodes héritées appelle une méthode redéfinie.
 - Une de ses méthodes héritées est appelée par une méthode redéfinie et utilise un paramètre qui est défini dans la méthode redéfinie.
- ❖ **CBM** (Coupling Between Methods) : mesure le nombre de méthodes nouvelles/redéfinie auxquelles toutes les méthodes héritées sont couplées. Un couplage quand on est de la donnée dans les conditions de la définition métrique IC.
- ❖ **AMC** (Average Method Complexity) : mesure la taille moyenne d'une méthode pour chaque classe. Taille d'une méthode est égale au nombre de codes binaires java dans la méthode.
- ❖ **CC** (Cyclomatic Complexity) : le nombre d'expressions suivantes trouvées dans le corps de la méthode if, while, foreach, cas, par défaut, goto, &&, ||, etc.

2.4. Les métriques utilisées par notre application

Dans ce tableau vous allez trouver toutes les métriques utilisées dans cette réalisation, et que nous avons choisies pour étudier notre approche.

Tableau 7: Les métriques utilisées

Métriques	Description	Catégorie	Pourquoi utilisée
WMC (Weighted methods per class)	le nombre des méthodes par classe ou simplement la somme de la complexité de ses méthodes	Une métrique de CK	
DIT (Depth of Inheritance Tree)	C'est la métrique fournit pour chaque classe une mesure des niveaux d'héritage haut de la hiérarchie d'objets. c'est la profondeur d'héritage	Une métrique de CK	
NOC (Number of Children)	cette métriques mesure simplement le nombre de descendants immédiats de la classe, tous simplement le nombre d'enfants d'une classe	Une métrique de CK	
NPM (Number of Public Methods)	La métrique NPM compte simplement toutes les méthodes dans une classe qui sont déclarées comme publiques. Elle peut être utilisée pour mesurer la taille d'une API fournies par un paquet.	Une métrique de QMOOD	Elle est accessible de l'extraire à partir du diagramme
DAM (Data Access Metric)	Cette métrique est le rapport du nombre d'attribue privé (protégé) au nombre total	Une métrique de Bansiya	

	d'attributs déclarés dans la classe. (Gamme 0-1).		de classe
MOA (Measure of Aggregation)	C'est le nombre de déclarations de données (champs de classe), dont les types sont des classes définies par l'utilisateur	Une métrique de Bansiya	
MFA (Measure of Functional Abstraction)	Cette métrique est le rapport du nombre de méthodes héritées par une classe et le nombre total de méthodes accessibles par des méthodes de membres de la classe. (Gamme 0-1).	Une métrique de Bansiya	
CAMC (Cohesion Among Methods of Class)	calcule le degré de parenté entre les méthodes d'une classe sur la base de la liste des paramètres des méthodes.	Une métrique de Bansiya	

2.5. Vision sur le site de PROMISE

PROMISE est l'un des plus grands dépôts de données de recherche SE (software engineering)

Le référentiel tera-PROMISE est un référentiel de données de recherches spécialisées dans les jeux de données de recherches en génie logiciel. Parrainé par la National Science Foundation, qui fournit un stockage gratuit et à long terme pour les données de recherche [24].

The screenshot shows the homepage of the PROMISE repository. At the top, there is a navigation bar with a home icon and the URL "/ repo / index.html". Below this is a welcome message: "Welcome to one of the largest repositories of SE research data". A paragraph explains the repository's goals: to provide free and long-term storage for research data and to be an open platform for finding and hosting datasets. A "Learn more" link is provided. Below the welcome message, there are two main sections: "Find research datasets" with a search icon and a "View categories" button, and "Contribute your data" with an upload icon and a "Learn how" button. On the left side, there is a "Data Categories" sidebar with a list of categories and their counts: Code Analysis (3), Defect (48), CK (33), McCabe & Halsted (13), Other (2), Dump (5), Effort (13), Cobol (1), Cocomo (3), Function Points Analysis (4), ISBSG (2), Personnel (1), Other (2), Green Mining (3), Issues (16), Model (4), MSR (3), Performance Predict (1), Requirements (5), NRP (2), Other (3), Search-Based SE (1), Social Analysis (1), Spreadsheet (4), and Other (21). Below the main content area, there is a "Featured dataset: EUSES" section with a description and a "View categories" button. To the right, there is a "Latest News" section with three news items: "May 05, 2015: The tera-PROMISE frontend has been rebuilt with Bootstrap.", "December 06, 2014: 80% of data migrated from Google Code site.", and "November 01, 2014: N.C. State approves the new repository space. One terabyte!". At the bottom, there is a "Most Recently Added Datasets (more here)" section with a table listing datasets: "Sourcerer ASTs" (Added on: May 25, 2015), "Project Planning" (Added on: May 25, 2015), "refactorcss" (Added on: May 01, 2015), and "MSRCB" (Added on: May 01, 2015).

Figure 37: Vision sur le site de PROMISE

Vous pouvez voir tous les ensembles de données dans les catégories énumérées sur la gauche et sur la page des catégories qui contient les différentes bases de données concernant les défauts, l'analyse de code, l'estimation de l'effort,....

opescience.us/repo/defect/ck/

tera-PROMISE Home About People Contact Contribute

Google Custom Search

repo / defect / ck / index.html

CK Metrics Total: 33

- For tutorial notes on the CK object-oriented metrics, see [here](#).
- For notes on the use of these metrics in defect prediction, see [this tutorial](#).
- For data sets that use the CK metrics, see below.

zuzel zuzel OO defect data	Added on: July 26, 2010
xerces xerces OO defect data	Added on: July 26, 2010
xalan xalan OO defect data	Added on: July 26, 2010
wspomaganiepi wspomaganiepi OO defect data	Added on: July 26, 2010

Data Categories

- Code Analysis 3
- Defect 48
- CK 33**
- McCabe & Halsted 13
- Other 2
- Dump 5
- Effort 13
- Cobol 1
- Cocomo 3

Figure 38: Les bases de données trouvées

Pour notre objectif nous nous intéressons à la catégorie defect, où nous trouvons un ensemble de bases de données tels que : zuzel, xerces, walan, ...etc.

Par exemple la base zuzel contient les informations sur le lien de téléchargement, les données et la référence:

zuzel

URL

- Latest version :
 - [zuzel](#)
- With change log:
 - [zuzel](#)

Change Log

When	What
July 26, 2010	Donated by Marian Jureczko

About the data

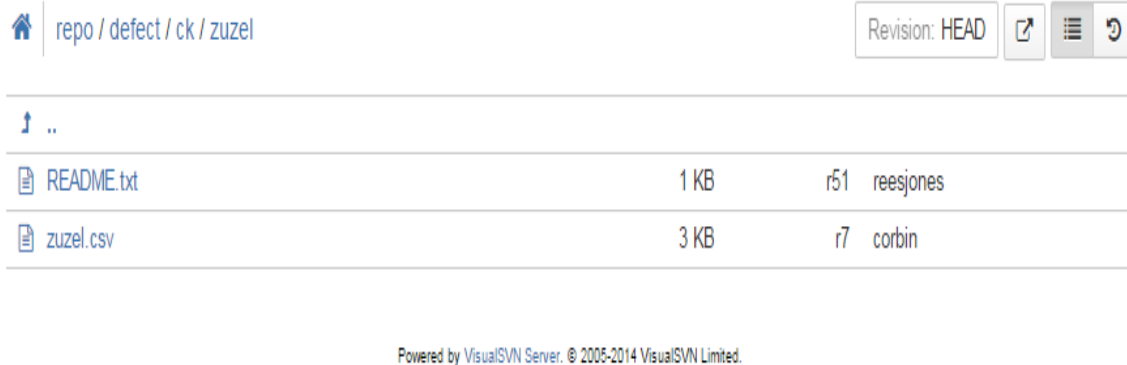
This dataset is one of the [Marian Jureczko Datasets](#).
This dataset uses the CK OO metrics.

Reference

Marian Jureczko and Lech Madeyski. 2010. Towards identifying software project clusters with regard to defect prediction. In Proceedings of the 6th International Conference on Predictive Models in Software Engineering (PROMISE '10). ACM, New York, NY, USA, , Article 9 , 10 pages. DOI=10.1145/1868328.1868342<http://doi.acm.org/10.1145/1868328.1868342>

Figure 39: Les informations disponibles pour chaque base de données

Et lorsqu'on clique sur le lien de téléchargement de données, cette page est affichée



repo / defect / ck / zuzel

Revision: HEAD

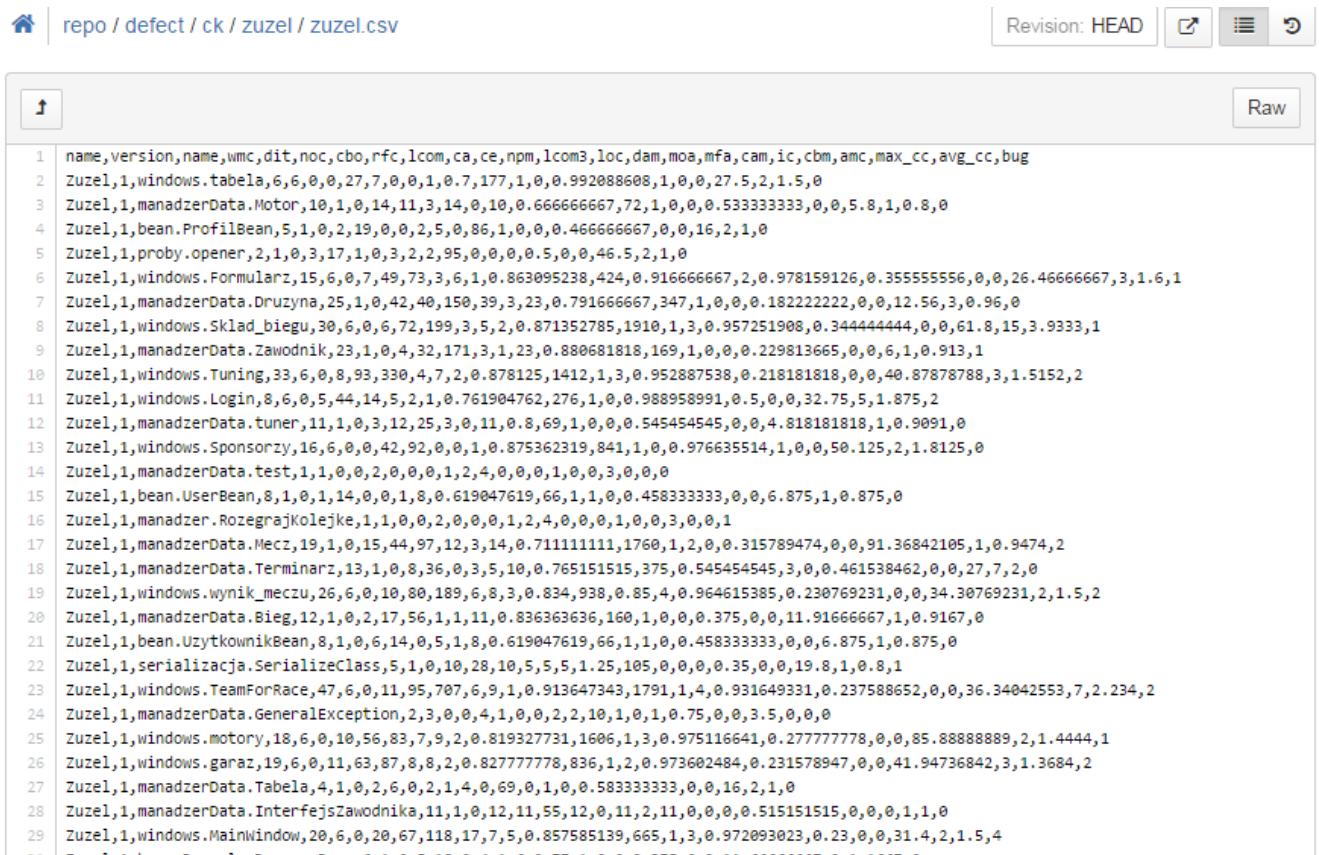
↑ ..

README.txt	1 KB	r51	reesjones
zuzel.csv	3 KB	r7	corbin

Powered by VisualSVN Server. © 2005-2014 VisualSVN Limited.

Figure 40:La fenêtre de téléchargement de la base de données zuzel

Et voilà la structure de la base de données sous forme de fichier CSV contenant le nom du projet, sa version et les métriques utilisées, plus un attribut pour le nombre de bug trouvée dans chaque projet.



repo / defect / ck / zuzel / zuzel.csv

Revision: HEAD

Raw

	name	version	name	wmc	dit	noc	cbo	rfc	lcom	ca	ce	npm	lcom3	loc	dam	moa	mfa	cam	ic	cbm	amc	max_cc	avg_cc	bug	
1	Zuzel	1	windows.tabela	6,6	0,0	27,7	0,0	1,0	7,177	1,0	0,9920888608	1,0	0,0	27,5	2,1	5,0									
2	Zuzel	1	manadzerData.Motor	10,1	0,14	11,3	14,0	10,0	6,666666667	72,1	0,0	0,533333333	0,0	0,5	8,1	0,8	0								
3	Zuzel	1	bean.ProfilBean	5,1	0,2	19,0	0,2	5,0	86,1	0,0	0,466666667	0,0	16,2	1,0											
4	Zuzel	1	proby.opener	2,1	0,3	17,1	0,3	2,2	95,0	0,0	0,5	0,0	46,5	2,1	0										
5	Zuzel	1	windows.Formularz	15,6	0,7	49,73	3,6	1,0	863095238	424,0	0,916666667	2,0	0,978159126	0,355555556	0,0	26,4	66666667	3,1	6,1						
6	Zuzel	1	manadzerData.Druzyna	25,1	0,42	40,150	39,3	23,0	791666667	347,1	0,0	0,182222222	0,0	12,56	3,0	96,0									
7	Zuzel	1	windows.Sklad_biegu	30,6	0,6	72,199	3,5	2,0	871352785	1910,1	3,0	0,957251908	0,344444444	0,0	61,8	15,3	9333,1								
8	Zuzel	1	manadzerData.Zawodnik	23,1	0,4	32,171	3,1	23,0	880681818	169,1	0,0	0,229813665	0,0	6,1	0,913	1									
9	Zuzel	1	windows.Tuning	33,6	0,8	93,330	4,7	2,0	878125	1412,1	3,0	0,952887538	0,218181818	0,0	40,87878788	3,1	5152,2								
10	Zuzel	1	windows.Login	8,6	0,5	44,14	5,2	1,0	761904762	276,1	0,0	0,988958991	0,5	0,0	32,75	5,1	875,2								
11	Zuzel	1	manadzerData.tuner	11,1	0,3	12,25	3,0	11,0	8,69	1,0	0,0	0,545454545	0,0	4,818181818	1,0	9091,0									
12	Zuzel	1	windows.Sponsorzy	16,6	0,0	42,92	0,0	1,0	875362319	841,1	0,0	0,976635514	1,0	0,50	125,2	1,8125	2								
13	Zuzel	1	manadzerData.test	1,1	0,0	2,0	0,0	1,2	4,0	0,0	1,0	0,0	3,0	0,0											
14	Zuzel	1	bean.UserBean	8,1	0,1	14,0	0,1	8,0	619047619	66,1	1,0	0,458333333	0,0	6,875	1,0	875,0									
15	Zuzel	1	manadzer.RozegrajKolejke	1,1	0,0	2,0	0,0	1,2	4,0	0,0	1,0	0,0	3,0	0,0	1										
16	Zuzel	1	manadzerData.Mecz	19,1	0,15	44,97	12,3	14,0	711111111	1760,1	2,0	0,315789474	0,0	91,36842105	1,0	9474,2									
17	Zuzel	1	manadzerData.Terminarz	13,1	0,8	36,0	3,5	10,0	765151515	375,0	0,545454545	3,0	0,461538462	0,0	27,7	2,0									
18	Zuzel	1	windows.wynik_meczu	26,6	0,10	80,189	6,8	3,0	834,938	0,85	4,0	0,964615385	0,230769231	0,0	34,30769231	2,1	5,2								
19	Zuzel	1	manadzerData.Bieg	12,1	0,2	17,56	1,1	11,0	836363636	160,1	0,0	0,375	0,0	11,91666667	1,0	9167,0									
20	Zuzel	1	bean.UzytkownikBean	8,1	0,6	14,0	5,1	8,0	619047619	66,1	1,0	0,458333333	0,0	6,875	1,0	875,0									
21	Zuzel	1	serializacja.SerializeClass	5,1	0,10	28,10	5,5	5,1	25,105	0,0	0,35	0,0	19,8	1,0	8,1										
22	Zuzel	1	windows.TeamForRace	47,6	0,11	95,707	6,9	1,0	913647343	1791,1	4,0	0,931649331	0,237588652	0,0	36,34042553	7,2	234,2								
23	Zuzel	1	manadzerData.GeneralException	2,3	0,0	4,1	0,0	2,2	10,1	0,1	0,75	0,0	3,5	0,0	0										
24	Zuzel	1	windows.motory	18,6	0,10	56,83	7,9	2,0	819327731	1606,1	3,0	0,975116641	0,277777778	0,0	85,88888889	2,1	4444,1								
25	Zuzel	1	windows.garaz	19,6	0,11	63,87	8,8	2,0	827777778	836,1	2,0	0,973602484	0,231578947	0,0	41,94736842	3,1	3684,2								
26	Zuzel	1	manadzerData.Tabela	4,1	0,2	6,0	2,1	4,0	69,0	1,0	0,583333333	0,0	16,2	1,0											
27	Zuzel	1	manadzerData.InterfejsZawodnika	11,1	0,12	11,55	12,0	11,2	11,0	0,0	0,515151515	0,0	0,0	1,1	0										
28	Zuzel	1	windows.Mainwindow	20,6	0,20	67,118	17,7	5,0	857585139	665,1	3,0	0,972093023	0,23	0,0	31,4	2,1	5,4								
29	Zuzel	1	bean.Powiazanie	6,1	0,5	10,4	1,0	0,0	77,1	0,0	0,375	0,0	11,6666667	0,1	167,0										

Figure 41:Exemple d'une base de données PROMISE (zuzel)

III. Outils de développement

1. WEKA

1.1 Définition



Weka (Waikato Environment for Knowledge Analysis) est une suite de logiciels libres (GPL) de datamining incluant de nombreux algorithmes d'apprentissages automatiques. Écrite en Java, développée à l'université de Waikato, Nouvelle-

Zélande. [28]

1.2. But ultime

Son but est de réaliser des tâches de fouille de données. Les algorithmes peuvent être appliqués directement à un ensemble de données ou appelés via un programme Java. En effet, Weka permet d'effectuer un prétraitement sur un ensemble de données, d'appliquer un algorithme d'apprentissage, et d'analyser les résultats et les performances d'un classificateur. Il est aussi bien adapté pour intégrer de nouveaux algorithmes d'apprentissage [28].

1.3. Version utilisée

La version utilisée dans ce document est la plus récente : la version 3.4.12.

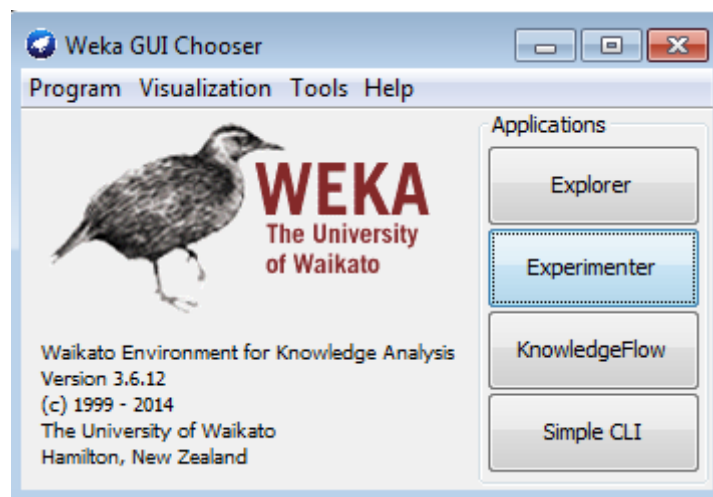


Figure 42: Environnement Weka

1.4. Composants de Weka

Weka possède plusieurs composants à savoir [28] :

- **Explorer** : ce module regroupe tous les packages importants de Weka à savoir le prétraitement, les algorithmes d'apprentissage, le groupement (clustering), les associations, la sélection des attributs et la visualisation.
- **Experimenter** : permet d'exécuter plusieurs algorithmes d'apprentissage en mode lot (batch) et de comparer leurs résultats.
- **KnowledgeFlow environment** : fournit les mêmes fonctionnalités que le composant "Explorer". Ces fonctionnalités sont représentées sous forme graphique et sont utilisées pour construire un schéma de flux de connaissances via une interface drag-and-drop.

Et dans notre propre recherche, nous avons concentré sur le module Explorer qui facilite l'accès aux différentes fonctionnalités de Weka, et permet à un utilisateur de réaliser certaines étapes de fouilles données.

Pour se faire, Explorer regroupe plusieurs packages tels que le package de prétraitement, les classificateurs, les clusters, les règles d'association, la sélection des attributs et un composant de visualisation. Comme la montre la figure ci-dessous.

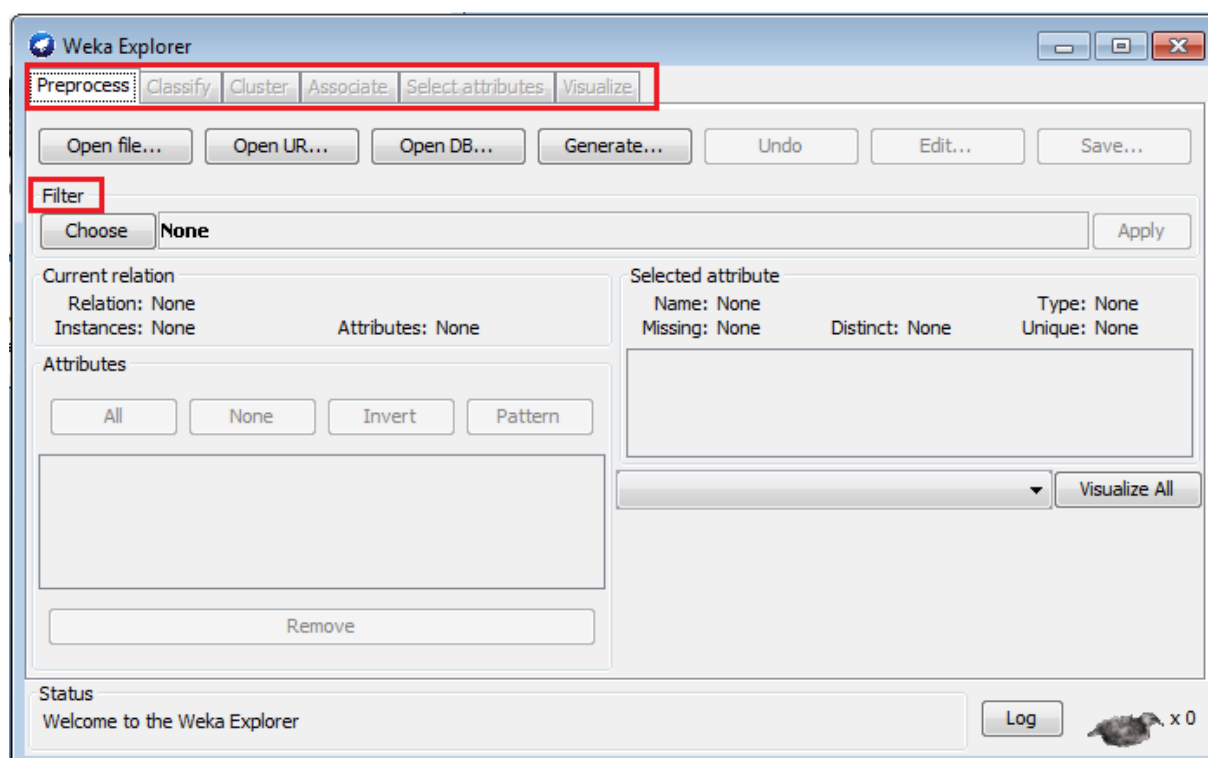


Figure 43: Capture d'écran du module Explorer de Weka

Cet outil permet donc de réaliser les expérimentations, en utilisant des moyens tels que les classificateurs, les clusters et les règles d'association, qui nécessitent au préalable un traitement des données. Dans notre recherche, on se concentre uniquement sur le prétraitement et l'application d'un algorithme d'apprentissage (classificateur).

Généralement, d'après la littérature, les données servant à l'expérimentation dans n'importe quel domaine sont stockées dans des fichiers Excel et le format des données utilisées par les tâches de fouille de données est "arff", plus le format utilisé par Weka est de type arff.

Avant d'appliquer donc les algorithmes d'apprentissage, ces données doivent être transformées en un fichier de données avec l'extension "arff" pour être lu.

Le format .ARFF est composé de :

- Commentaires : %
- Relation : @RELATION [nom de la relation]
- Attribut : @ATTRIBUTE [nom de l'attribut] [type]
- Données : @DATA

[Valeur], [valeur],..., [valeur]

[Valeur], [valeur],..., [valeur]

...

[Valeur], [valeur],..., [valeur]

Pour se faire nous avons utilisé ce site «<http://slavnik.fe.uni-lj.si/markot/csv2arff/csv2arff.php>» qui nous permet de convertir un fichier d'extension csv (car les fichiers de la base de données PROMISE sont de types csv) à un fichier d'extension arff [29].



Figure 44:Le site de conversation CSV à ARFF

Dans le package de prétraitement, nous trouvons :

- Dans la partie supérieure, les trois possibilités d'ouverture d'un fichier de données d'apprentissage qui sont "Open file", "Open URL" et "Open DB";
- Le choix du filtre en cas de nécessité, d'information sur la relation et l'attribut sélectionné, d'une partie où tous les attributs sont affichés et une zone de visualisation en forme d'histogrammes de la distribution de l'attribut sélectionné.

La figure ci-dessous nous montre le contenu de la fenêtre de prétraitement après chargement du fichier de données de l'expérimentation via l'activation de l'option "Open file".

Le fichier utilisé est « xalan-2.arfff ».

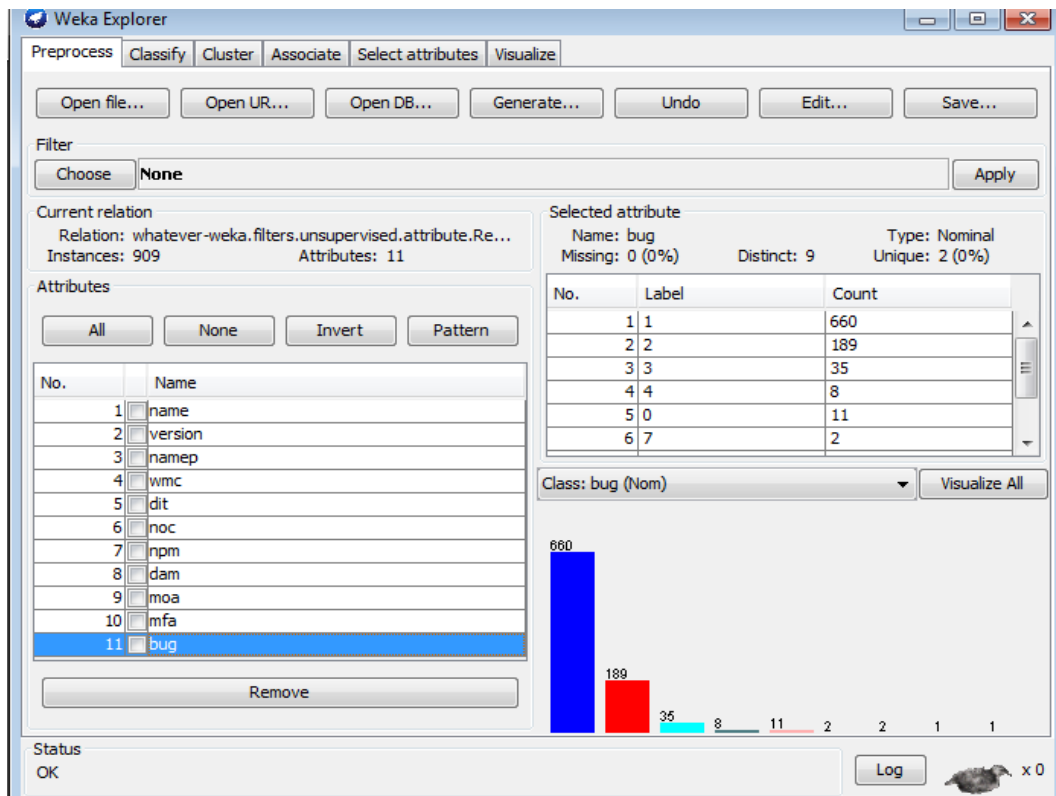


Figure 45:Chargement de l'ensemble des données

Dans la partie gauche, nous avons dans le premier cadre les informations sur les données de l'expérimentation chargées, ayant 909 instances et 11 attributs (10 variables indépendantes et la variable dépendante class). Tous les attributs sont affichés dans le cadre attributs. La sélection de la variable class dans la liste des attributs fait apparaître dans la partie droite les informations sur cette variable telle que le nom et le type. Avec la sélection de la variable à visualiser dans la partie droite, un histogramme des valeurs prises est affiché.

L'apprentissage supervisé est disponible dans l'onglet **Classify**. Une fois le fichier chargé, ce dernier onglet est activé et plusieurs algorithmes d'apprentissage supervisé nous sont proposés.

Un extrait de tous les algorithmes d'apprentissage est présenté dans la figure ci-dessous.

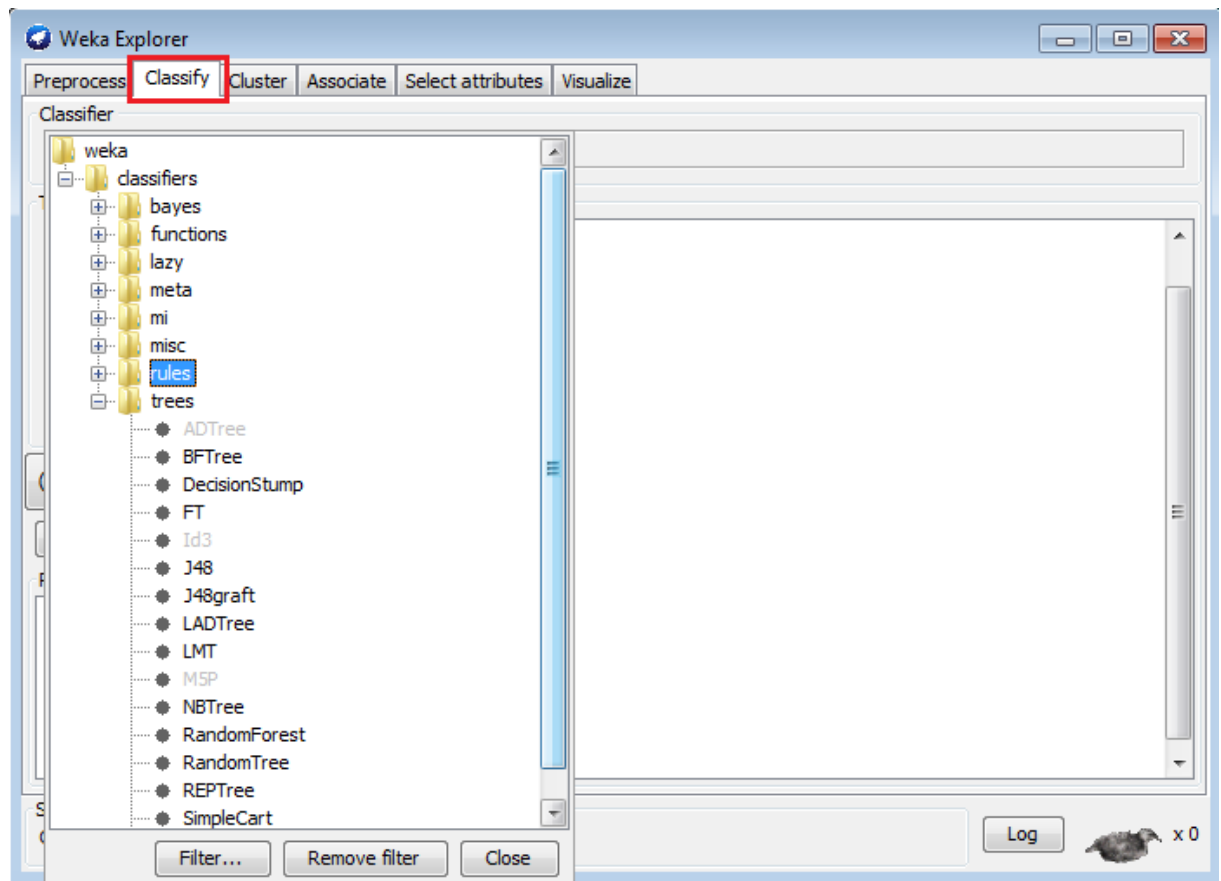


Figure 46: Familles d'algorithmes d'apprentissages automatiques Supervisés

La figure suivante montre les résultats obtenus par l'application de l'algorithme J48 (une implantation de C4.5) sur l'ensemble de données « xalan-2.arff ».

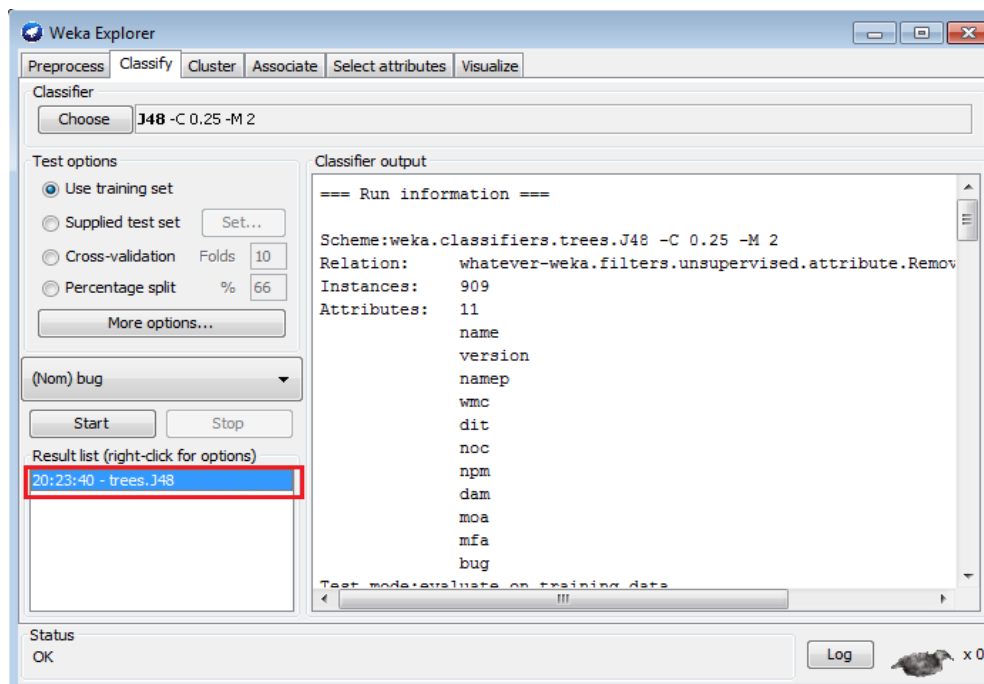


Figure 47:Résultats de l'algorithme J48

Les résultats sont structurés en trois volets :

- un volet résultats sommaires : donne le nombre total d'instances classifiées correctement et incorrectement, la valeur de kappa, l'erreur absolue moyenne, l'erreur racine carrée moyenne, l'erreur relative absolue et l'erreur racine carrée relative.
- un volet résultats par classes : fournit le taux d'instances classifiées correctement et incorrectement via les taux TP (true positif) et FP (false positif), la précision et d'autres informations statistiques.
- un volet matrice de confusion : donne plus de précision concernant le nombre d'instances correctement et incorrectement classifiées correspondants aux taux TP et FP pour chaque classe.

2. NetBeans

2.1. Définition



C'est un environnement de développement intégré (EDI), en plus de Java, NetBeans permet également de supporter différents autres langages, comme C, C++, JavaScript, XML, Groovy, PHP et HTML de façon native ainsi que bien d'autres. Il comprend toutes les caractéristiques d'un IDE moderne (éditeur en couleur, projets multi-langage, refactoring, éditeur graphique d'interfaces et de pages Web) [30].

2.2. But ultime

Son but est d'être le meilleur EDI/IDE pour le développement Java/JEE notamment pour les applications web [30].

2.3. Version utilisée

La version utilisée dans ce document est: la version IDE 8.0.2.



Figure 48:Version de NetBeans utilisé

3. Vue global sur l'application

3.1. Objectif

Pour mettre en place notre étude, nous avons réalisé une application qui est analogue au fonctionnement de Weka.

L'interface a pour but :

D'appliquer certains algorithmes pour l'exploration de données

- Clustering, classification, régression, etc.
- Analyse de résultats
- Évaluation de performances, comparaison d'algorithmes

Les classifieurs sont des modèles pour prédire des valeurs nominales ou numériques. Les algorithmes de classification inclus dans notre application sont:

- Arbres de décision
- Machine à vecteurs de support (SVM)
- Perceptron multi-couche
- Réseau bayésien

3.2. Description

Notre application se compose de 3 grands axes :

1. Classification avec une des méthodes non supervisées (k-means,c-means,dbscan,kohonen)
 - Chargement des données (.data)
 - Choix d'une méthode
 - Visualisation des résultats
2. algorithmes de classification supervisée (arbre de décision, SVM, PM, RB)
 - Chargement des données arff
 - Application d'un des algorithmes de classification
 - Résultat
3. Prédiction de taux de défaut d'un projet
 - Chargement de fichier d'apprentissage
 - Chargement de fichier test
 - Obtention d'un fichier de sortie contenant le résultat de prédiction

4. Capture d'écran

L'Interface d'accueil :

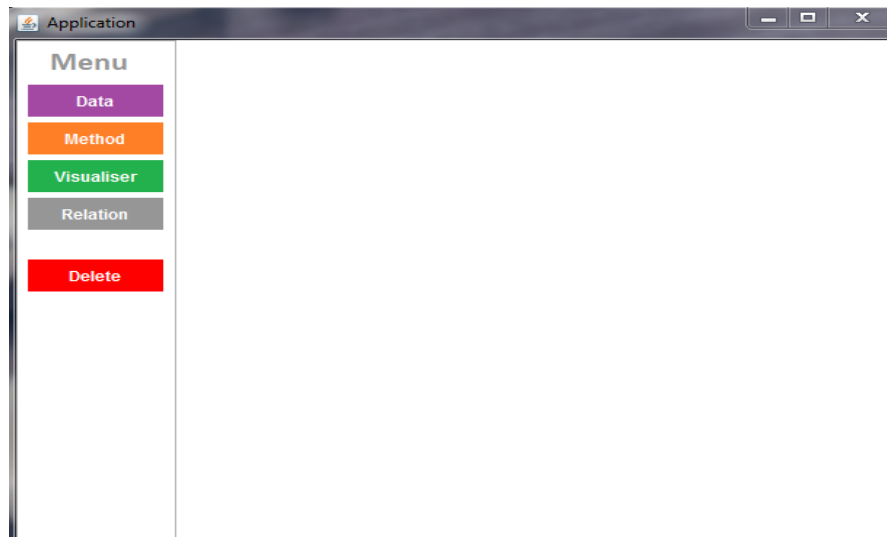


Figure 49:Fenêtre principale de l'application

Choix d'une méthode :

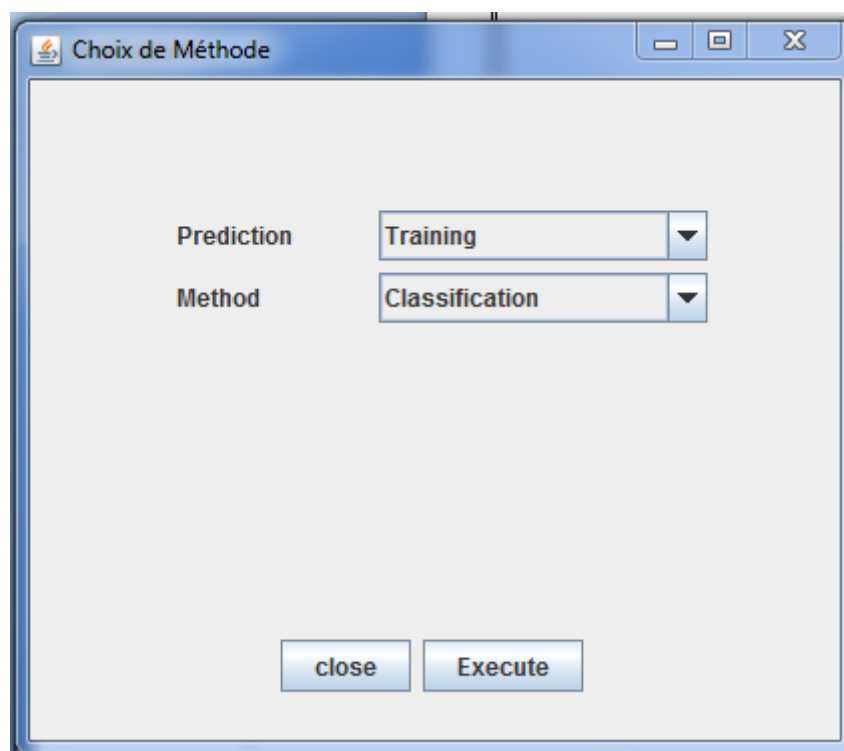


Figure 50:Choix de méthode d'apprentissage

Une petite description sur la méthode choisie apparaît, ainsi que la fenêtre de choix de l'emplacement de fichier arff.

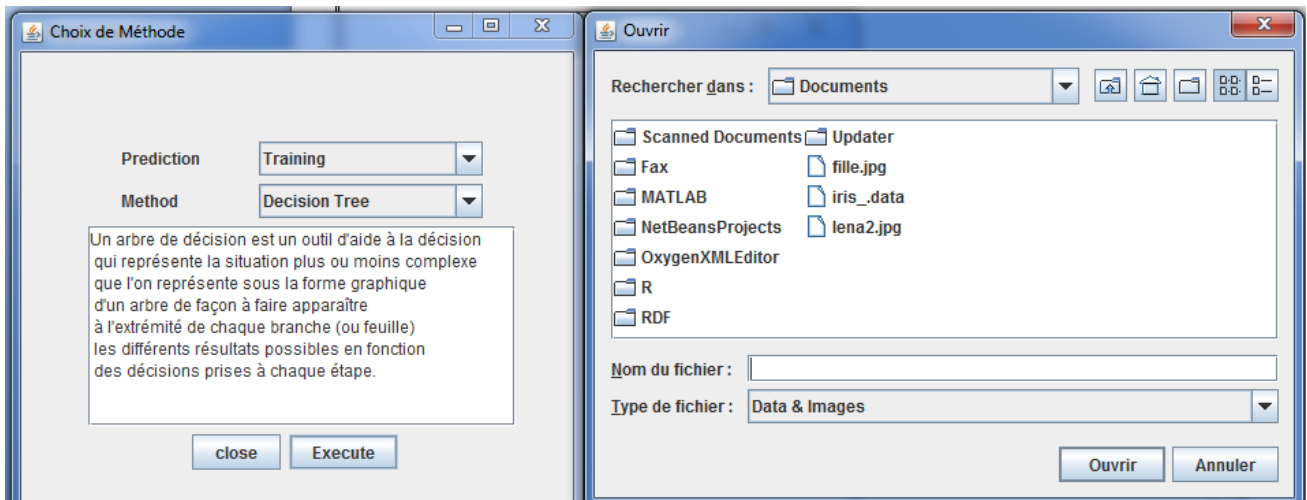


Figure 51:Fenêtre d'une étape de l'application

Est le résultat s'affiche au niveau d'une simple fenêtre comme le montre la figure suivante :

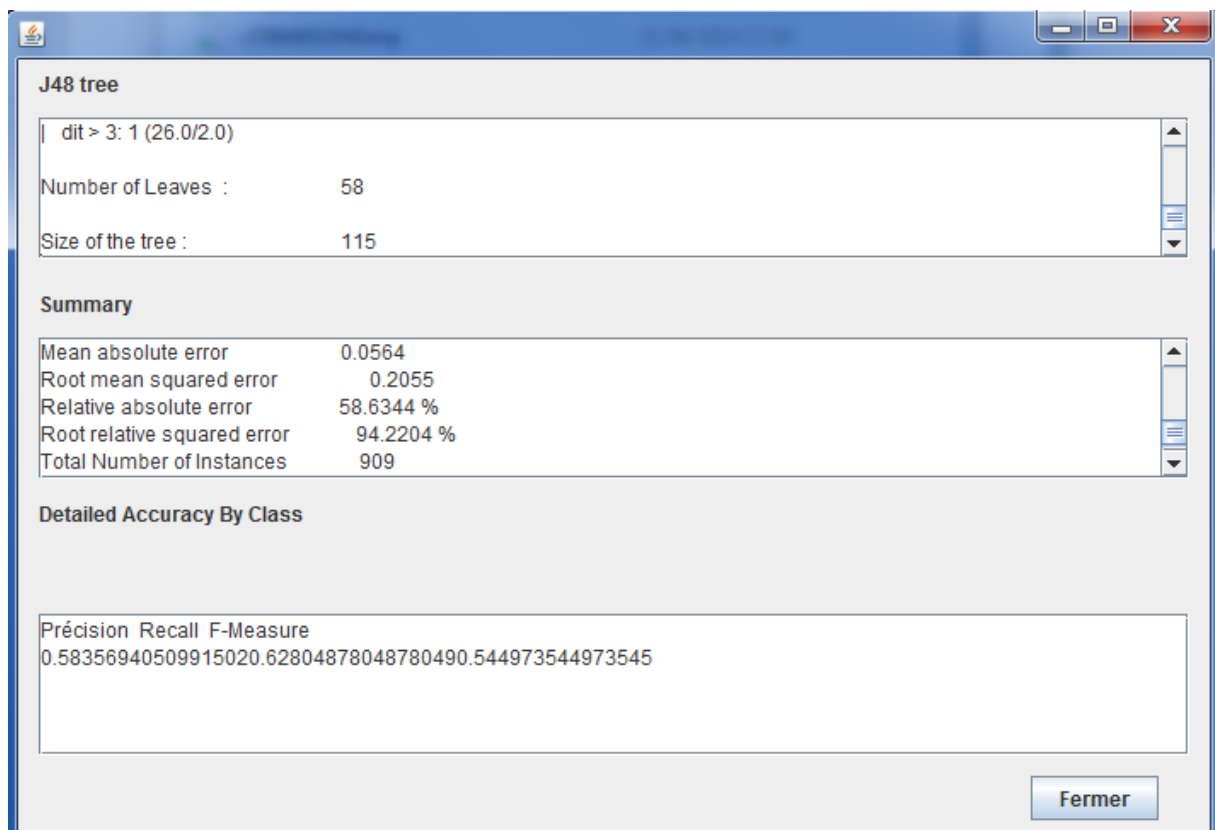


Figure 52:Exemple de résultat avec les arbres de décision

5. Étude de convergence des méthodes utilisées

Dans cette partie nous essayons d'étudier la convergence des 3 méthodes : arbre de décision, SVM, naïve bayes, ces techniques sont utilisées comme des améliorations à prévoir des données inconnues ou disparues, pour cet objectif nous avons utilisé :

- Un fichier pour l'**apprentissage**, contenant les différentes mesures de métriques de certains projets, ainsi que le nombre de bugs trouvés pour chaque projet.

- Un formulaire pour remplir les différentes métriques du projet en question.

La comparaison de la précision des différents modèles est une tâche importante pour évaluer l'exactitude des algorithmes de datamining.

Le tableau suivant résume Les résultats obtenus

Tableau 8:Résultat d'étude de convergence de chaque méthode

Valeur original	Arbre de décision	Réseau bayes	SVM
1	1	1	1
1	1	0	1
1	1	1	1
1	2	2	1
1	1	0	1
1	1	1	1
2	1	2	1
2	2	2	1
3	1	4	1
2	1	3	1
Precision	0.911	0.718	0.992

L'étude de la précision de chaque méthode par rapport aux autres nous donne une vision sur la convergence vers le meilleur résultat. Comme résultat nous remarquons que les arbres de décision, SVM, montre leurs efficacités au niveau de la prédiction.

IV. Conclusion

Dans ce chapitre nous avons présenté la base de données choisie et la raison de ce choix , nous avons aussi présenté des petites descriptions sur les métriques utilisées par notre base de données et plus spécifiquement les métriques choisies pour notre application, nous avons aussi présenté une vue globale sur notre application, et finalement nous avons étudié la convergence des méthodes d'apprentissage automatique supervisé à savoir les arbres de décision, les SVM, les réseaux de neurone et les réseaux bayésiens, nous avons observé que les arbres de décision, les SVM, montrent leurs efficacité au niveau de la prédiction.

Conclusion et perspectives

La motivation principale de ce travail de recherche consiste à prédire les bugs des logiciels orientés objets dans les phases amants de leur conception afin d'améliorer la qualité, d'éviter les échecs lors de l'exécution du logiciel, d'éliminer les raison principales de ces bugs et pour identifier une démarche d'assurance de la qualité dès le début de développement.

Afin de bien atteindre notre objectif, nous avons tout d'abord choisi la base de données **PROMISE** (Predictor Models In Software Engineering), dans laquelle nous avons sélectionné **des métriques pour les logiciels orientées objets**, que nous pouvons extraire à partir **de diagramme de classes**, qui est considéré comme un élément crucial dans les phases amants de conception. Nous avons utilisé des **méthodes DataMining** comme **technique d'aide à la décision**. Parmi ces méthodes nous trouvons : les arbres de décision, les réseaux bayesiens, les SVMs et les réseaux de neurones.

En ce qui concerne la validation pratique, nous avons développé une application qui permet de faire la classification avec les différentes méthodes de DataMining et la prédiction des nombres des bugs d'un projet.

En effet nos activités futures incluent :

- L'exploitation d'autre diagramme de l'UML (Unified Modelling Langage) tel que les diagrammes de cas d'utilisation, les diagrammes de séquences, etc. afin de définir d'autres métriques.
- sélection des métriques pour la prédiction du coût
- amélioration de notre application d'une manière très professionnelle, de telles sortes qu'un utilisateur peut extraire les mesures souhaitées à partir du diagramme de classes d'une manière automatique.

Au terme de ce projet, notre sentiment est très positif; nous avons effectivement acquis de nombreuses connaissances et des nouvelles compétences.

Une bonne expérience à renouveler !

Références

- [1] UML Introduction au génie logiciel et à la modélisation ; Delphine Longuet.
- [2] <http://www.commentcamarche.net/contents/473-cycle-de-vie-d-un-logiciel>.
- [3] <http://mariepascal.delamare.free.fr/IMG/pdf/coursQualite.pdf>.
- [4] Qualité du logiciel et métriques Hiver ; Chapitre 3 – Qualité du logiciel, Houari Sahraoui ; 2006.
- [5] La mesure des modèles par les modèles : une approche générative ; Martin Monperrus ; 2011.
- [6] analyse des indicateurs nécessaires au processus d'évaluation du software pour l'amélioration de la qualité du logiciel; Lazhar OURLIS ; 2012.
- [7] Les métriques (McCabe, Halstead) ; l'index de maintenabilité et leur influence sur la qualité, Klaus LAMBERTZ ; Verifysoft Technology.
- [8] http://www.verifysoft.com/fr_halstead_metrics.html.
- [9] Génie Logiciel Principes et Techniques, Pierre Gérard; 2008.
- [10] Indicateur de qualité pour les systèmes orientés objet : vers un modèle unifiant plusieurs métriques; FADEL TOURE ; Décembre 2007.
- [11] http://fr.wikipedia.org/wiki/UML_%28informatique%29#Utilit.C3.A9_d.27UML.
- [12]http://fr.wikipedia.org/wiki/Diagramme_de_classes.
- [13] A Survey of Metrics for UML Class Diagrams; Marcela Genero, Mario Piattini, Coral Calero; Novembre-Décembre 2005.
- [14] http://fr.wikipedia.org/wiki/Aide_à_la_décision.
- [15] Utilisation des outils d'aide à la décision dans la gestion des mégasites ; décembre 2006.
- [16] Projet Datamining ; JUIN 2004.
- [17] Data Mining ; Techniques d'extraction des connaissances ; Georges El Helou et Charbel Abou khalil ; février 2004.
- [18] Neural network a comprehensive foundation; Prentice-Hall; 1994.
- [19] fr.Wikiversité.org/wiki/arbres_de_décision.

- [20] Les Méthodes d'Apprentissage Supervisé (Arbre de décision, KPPV, SVM); Fakhour Mohammed, Zayna Oufqir, Rachida Moulay Taj, Salma Filali ; 2015.
- [21] Les séparateurs a vaste marge Bi-classes ; KHELLAT-KIHEL Souad ; 2012.
- [22] Les Réseaux Bayésiens, Abdelhamid El hassani; Nacer Harti , Yassine el Malyh ; 2015.
- [23] On Software Engineering Repositories and their Open problems; Daniel Rodriguez, Israel Herraiz, Rachel Harrison.
- [24] <http://promisedata.org/2014/>.
- [25] http://gromit.iar.pwr.wroc.pl/p_inf/ckjm/metric.html.
- [26] <http://openscience.us/repo/defect/ck/>.
- [27] http://fr.wikipedia.org/wiki/Weka_%28apprentissage_automatique%29.
- [28] Document d'utilisation : Environnement Weka ; H. Yazid ; 2006.
- [29] <http://slavnik.fe.uni-lj.si/markot/csv2arff/csv2arff.php>.
- [30] <http://fr.wikipedia.org/wiki/NetBeans>.