UNIVERSITE SIDI MOHAMED BEN ABDELLAH
FACULTE DES SCIENCES ET TECHNIQUES FES
DEPARTEMENT D'INFORMATIQUE

Projet de Fin d'Etudes

Master Sciences et Techniques Systèmes Intelligents et Réseaux

Le Datamining dans la prédiction du coût de logiciel

Lieu de stage : Laboratoire Systèmes Intelligents et Applications

Réalisé par :

EL ISSAOUI Naoufal

Encadré par :

Pr. Loubna LAMRINI

Soutenu le 13/06/16 devant le jury composé de :

Pr. Loubna **LAMRINI**

Pr. Fatiha **MRABTI**

Pr. Saïd **NAJAH**

Pr. Mohamed **OUZARF**

Année Universitaire : 2015-2016

Dédicaces

A :

Mes très chers parents et grands-parents :

J'ai reçu de vous de nombreuses leçons avec beaucoup d'amour et de tendresse dont le mérite, les sacrifices et les qualités humaines m'ont permis de vivre ce jour. Rien au monde ne pourrait vous compenser pour vos efforts, je veux bien dire combien vous m'êtes chers. Que le Bon Dieu vous sauvegarde pour moi et vous bénisse.

Mes sœurs et mon frère,

Qui m'ont bien soutenu moralement durant la préparation du présent mémoire, j'espère être à la hauteur de vos espérances.

Mes amis,

Je dédie également ce travail à tous mes amis, et à tous ceux qui m'ont bien aidé de près ou de loin durant la période de mon stage et pendant la réalisation de ce travail.

Remerciements

Après Dieu, je tiens à adresser mes sincères remerciements à mon encadreur Mme le professeur :

Loubna LAMRINI

Je vous remercie pour votre accueil et vos conseils. Veuillez trouver ici l'expression de mes gratitudes pour tous les conseils et remarques constructives que vous m'aviez adressées tout le long de ce travail.

Aux membres du jury :

Vous me faites un grand honneur en acceptant de juger ce travail. Certes vos qualités scientifiques et pédagogiques vont m'éclaircir la bonne voie qui mène au domaine de la recherche scientifique.

Mes profondes gratitudes s'orientent aussi vers :

*Toute l'**équipe pédagogique** du département informatique, après ces longues années d'études pour ses judicieux conseils et son support permanent dont le fruit est presque mûr et s'apprête à être cueillit, toutefois sans oublier de remercier sans exception tout le personnel administratif et pédagogique de la FST.*

Résumé

L'estimation du coût de logiciel peut être décrite comme le processus de prédiction de l'effort nécessaire pour développer un logiciel. Pour avoir une bonne estimation, plusieurs méthodes d'estimation ont été proposées. Mais leur utilisation, dès les premières phases, reste très difficile et n'offre pas une exactitude acceptable.

Pour notre objectif, nous nous intéressons à estimer l'effort d'un projet logiciel en choisissant des métriques pouvant être extraites dès les premières phases d'un projet, afin d'appliquer des méthodes datamining, et de comparer les résultats obtenus.

MOTS-CLES : estimation du coût de logiciel, métriques, méthodes datamining.

Abstract

The estimated cost of software can be described as the prediction process of the effort required to develop the software. In order to get a good estimate, several estimation methods have been proposed. But their use in the earlier stages is still very difficult and does not offer an acceptable accuracy.

In regards with our aim , we are interested in estimating the effort of a software project by choosing metrics that can be retrieved from the earlier stages of a project to use data mining methods and compare the results.

KEYWORDS : cost estimation software, metrics, data mining methods.

Sommaire

Liste des figures

Liste des tableaux

Introduction générale

L'estimation des coûts de développement de logiciels constitue toujours une tâche complexe à accomplir et une préoccupation majeure pour les gestionnaires de projets logiciels. Améliorer la précision des estimations fournies leur permettra un contrôle effectif du temps et du budget durant tout le cycle de développement du logiciel.

Pour ce faire, plusieurs modèles d'estimation ont été développés et utilisés. Cependant, le manque de données précises et/ou certaines sur le logiciel au début de son développement d'une part, et la diversité des facteurs influençant le coût d'autre part, ont rendu ces modèles imprécis et peu fiables.

Notre objectif dans ce travail, s'appuie sur la prédiction du coût de logiciels, on choisissant des métriques qui peuvent être obtenus dans les phases initiales du développement et en appliquant plusieurs méthodes Datamining afin d'estimer l'effort du projet logiciel.

Pour réaliser ce travail, nous avons utilisé la base de données qui contient l'historique de 75 projets **décrits avec des métriques** pouvant être obtenus dans les phases initiales du développement d'un logiciel, et nous avons appliqué les méthodes datamining (**perceptron multicouche**, **SVM**, **Régression linéaire**, **arbre de décision**) afin d'estimer l'effort d'un projet logiciel.

Le rapport est structuré sur 4 chapitres, le premier est destiné à présenter les différentes méthodes d'estimation du coût, le 2ème est consacré à présenter quelques métriques logiciels, ensuite, pour le 3ème nous présentons les méthodes datamining utilisées pour notre étude et enfin la réalisation de l'application, et pour finir nous terminerons avec une conclusion et des perspectives.

Chapitre 1 :

Méthodes d'estimation de coût

I) Introduction

L'estimation des coûts de développement de logiciels est une tâche très complexe dans la gestion d'un projet logiciel. Au cours de ces dernières années, le logiciel est devenu le composant le plus coûteux des projets de systèmes informatiques [1]. Cependant, un certain nombre de modèles ont été proposé afin d'estimer le coût d'un nouveau projet logiciel, mais le grand problème qui persiste, c'est que les données du projet disponible dans les premières étapes du projet est souvent incomplètes, incohérentes, incertaine et imprécise [2].

L'estimation de l'effort nécessaire pour un nouveau projet est estimée en prenant les détails du nouveau projet en compte. Le projet spécifique est ensuite comparée à un ensemble de données historiques (un ensemble de projets antérieurs) contenant des mesures de paramètres pertinents (par exemple, la taille, la langue utilisée, et de l'expérience du développement équipe) et l'effort de développement associé [3].

La majeure partie du coût de développement de logiciels est due à l'effort humain, et la plupart des méthodes d'estimation des coûts mettent l'accent sur cet aspect et donnent des estimations en termes de **personnes-mois (**L'effort de développement d'un logiciel représente le nombre de personnes ainsi que le nombre de mois nécessaires pour achever un projet. Par exemple, si 10 personnes ont travaillé pendant 5 mois dans un projet, l'effort de développement de ce projet est alors 50 Homme-mois) [4].

Par conséquent, si on connaît les salaires mensuels de ces intervenants dans le projet, son coût est tout simplement la somme des salaires multipliés par le nombre de mois de travail. Ceci n'implique donc que les trois attributs : **effort**, **temps** et **coût**, ils sont étroitement liés [1].

- ✓ **Effort** (généralement en mois-personnes).
- ✓ **Durée du projet** (dans le temps de calendrier).
- ✓ **Coût** (en dollars par exemple).

Des estimations précises des coûts de logiciels sont essentielles pour les développeurs et les clients. Ils peuvent être utilisés pour générer la

demande de propositions, les négociations contractuelles, la planification, le suivi et le contrôle.

La sous-estimation des coûts peut entraîner la gestion à approuver les systèmes proposés qui peuvent dépasser leurs budgets, avec des fonctions sous-développées et de mauvaise qualité, comme elle peut entraîner l'échec d'achever le projet dans les délais déterminés. La surestimation peut entraîner trop de ressources engagées dans le projet, ou pendant l'exécution du contrat d'appel d'offres, comme elle peut entrainer la perte du contrat, et par conséquence la perte d'emplois [1].

II) Processus d'estimation

Le processus d'estimation du coût réel comporte sept étapes [1] comme suit :

1. Établissement des objectifs de coûts-estimation.

2. Génération d'un plan de projet pour les données et les ressources nécessaires.

3. Fixer les exigences logicielles.

4. Travailler le plus possible de détails sur le système du logiciel.

5. Utiliser plusieurs techniques d'estimation des coûts indépendants.

6. Comparer les différentes méthodes d'estimation et itérer le processus d'estimation.

7. Après le démarrage du projet, suivi de son coût réel et l'état de son progrès, et après les résultats de rétroaction de la gestion de projet.

→Peu importe le modèle d'estimation sélectionnée, les utilisateurs doivent faire attention à ce qui suit pour obtenir de meilleurs résultats :

- La couverture de l'estimation (certains modèles génèrent l'effort pour le cycle de vie complet, tandis que d'autres ne prennent pas en considération cet effort pour la phase de prescription).

- Étalonnage et hypothèses du modèle.
- Sensibilité des estimations aux différents paramètres du modèle.
- Type de l'estimation par rapport au coût réel.

III) Les modèles algorithmiques d'estimation de coût

Les modèles algorithmiques (ou paramétriques) d'estimation des coûts de développement de logiciels se basent essentiellement sur une équation centrale exprimant l'effort en fonction d'un certain nombre de variable considérées comme étant les principales conductrices du coût.

Tout modèle algorithmique est à la forme :

Effort = f (x1, x2, ..., xn)

Dont :

L'effort : Représente l'effort de développement.

Fonction d'application f : Représente la relation reliant l'effort aux facteurs conducteurs des coûts.

(x1,…, xn) : Sont les variables prédictives représentant les facteurs conducteurs des coûts.

1) Modèles basés sur la régression linéaires

Ces modèles adoptent une équation centrale linéaire de la forme [1] :

$$\text{Effort} = a_0 + \sum_{i=1}^{n} a_i x_i$$

Où les coefficients a1, ..., an sont choisis pour répondre au mieux aux données du projet terminé. La mise au point de tels modèles est souvent

accomplie en utilisant les techniques statistiques de régression linéaire simple ou multiple selon le nombre de variables xi**. La régression linéaire** consiste à analyser un ensemble de données sur des projets logiciels déjà achevés afin de reconstruire la relation, supposée être linéaire, exprimant l'effort (variable dépendante) en fonction des variables xi (variables indépendantes).

2) COCOMO (Constructive Cost Model)

COCOMO est un acronyme pour COnstructive COst MOdel. C'est une méthode pour estimer le coût d'un projet logiciel dans le but d'éviter les erreurs de budgets et les retards de livraison, qui sont malheureusement habituels dans l'industrie de développement logiciel [7].

Il est constitué de trois modèles :

- **La base COCOMO :**

Il s'appuie uniquement sur la taille estimée du logiciel et sur le type de logiciel à développer.

Des familles différentes sont proposées pour trois types de projets :
Effort = B * (kLOC) ^C

Projet organique B=2.4, C=1.05
- ✓ Petite équipe
- ✓ Environnement connu
- ✓ Expérience avec ce type d'application

Projet embarqué B=3.6, C=1.2
- ✓ Projet complexe
- ✓ Contrainte sévère imposée par l'environnement

Projet semi-détaché B=3.0, C=1.12
- ✓ Expérience moyenne
- ✓ Projet de grande taille

Avec **B** et **C** : (généralement simples) qui sont des fonctions d'autres facteurs de coût.

KLOC : La taille du code.

- **Intermédiaire COCOMO :**

Dans le COCOMO intermédiaire, l'estimation de l'effort nominal est obtenue en utilisant la fonction de puissance avec trois ensembles de {a, b}, avec un coefficient étant légèrement différent de celui de la base COCOMO :

Projet organique B=3,2, C=1.05
- ✓ Petite équipe
- ✓ Environnement connu
- ✓ Expérience avec ce type d'application

Projet embarqué B=3.0, C=1.15
- ✓ Projet complexe
- ✓ Contrainte sévère imposée par l'environnement

Projet semi-détaché B=2,8, C=1.2
- ✓ Expérience moyenne
- ✓ Projet de grande taille

- **Expert COCOMO :**

Il inclue toutes les caractéristiques du modèle intermédiaire avec une estimation de l'impact de la conduite des coûts sur chaque étape du cycle de développement : définition initiale du produit, définition détaillée, codage, intégration. De plus, le projet est analysé en termes d'une hiérarchie : module, sous système et système. Il permet une véritable gestion de projet, utile pour de grands projets.

COCOMO II : La différence la plus importante par rapport aux premiers modèles COCOMO est que l'exposant b change selon la souplesse de développement, l'architecture ou le risque de résolution, la cohésion d'équipe, et le processus maturité.

COCOMO II est constitué en fait de trois modèles [6] :

- **Modèle de composition d'application**

Ce modèle est utilisé pour les projets fabriqués à l'aide des toolkits d'outils graphiques. Il est basé sur les nouveaux « Object Points ».

- **Modèle avant-projet**

Modèle utilisé pour obtenir une estimation approximative avant de connaître l'architecture définitive. Il utilise un sous ensemble de facteurs de productivité (Cost drivers). Il est basé sur le nombre de lignes de code ou les points de fonction non ajustés.

- **Modèle post-architecture**

Il s'agit du modèle le plus détaillé de COCOMO II. A utiliser après le développement de l'architecture générale du projet. Il utilise des facteurs de productivité (Cost drivers) et des formules.

Avantage :

-COCOMO reste la référence en matière d'estimation détaillée des coûts et surtout de la ventilation de ces coûts suivant les phases des projets.

- Les estimations de COCOMO sont d'autant plus fiables, que les paramètres du projet sont bien connus, c'est-à-dire que les projets précédents ont servi pour étalonner ces paramètres.

- COCOMO à l'avantage d'être ouvert. Les données de calibrage, les formules et tous les détails des définitions sont disponibles. La participation à son développement est encouragée.

Inconvénients :

Ils résident dans la nécessité d'avoir une estimation du nombre de lignes du logiciel (LOC). Cette taille du logiciel n'est connue qu'à la fin de la réalisation et le problème de son estimation reste entier.

3) Points de fonction

C'est une méthode destinée à évaluer la taille d'un projet indépendamment de la technologie utilisée.

Le principe de cette méthode est de faire une estimation à partir d'une description externe du futur système, de ses « fonctions » ou composants « fonctionnels».

On identifie **cinq** types avec **trois** degrés de complexité. À chaque type et chaque degré est affecté un nombre de «points », ce qui permet de calculer, pour un projet donné, son poids en *points de fonction.*

- Pour les Cinq types d'entités fonctionnelles :
  - Deux sont relatifs à l'aspect statique d'un système d'information (GDI et GDE).
  - Trois à l'aspect dynamique (ENT, SOR et INT).

3-1) GDI (Groupe de Données Interne) :

- Relatif à l'aspect statique du Système d'Information.

- Groupe de données logiquement liées, ou groupe de paramètres de contrôle, et identifiables par l'utilisateur.

- Ces données sont mises à jour et utilisées à l'intérieur de la frontière de l'application.

3-2) GDE (Groupe de Données Externe) :

- Relatif à l'aspect statique du Système d'Information.

- Groupe de données logiquement liées, ou groupe de paramètre de contrôle, et identifiables par l'utilisateur.

- Ces données sont utilisées par l'application, mais mises à jour par une autre application.

- Le GDE est un GDI dans un autre domaine.

3-3) ENT (Entrée) :

- Relatif à l'aspect dynamique du Système d'Information.
- Ce sont les données, ou les paramètres de contrôle, qui entrent dans l'application considérée. Ces entrées maintiennent un ou plusieurs GDI, initialisent ou contrôlent un traitement, et font l'objet d'un traitement unique.
- Une Entrée correspond donc à un écran de saisie, ou à une réception de données.
- A Chaque GDI doit correspondre au moins une entrée, permettant sa mise à jour.

3-4) INT (Interrogation) :

– C'est une fonction élémentaire, qui a pour résultat l'extraction de données. La demande d'interrogation peut être saisie ou provenir d'une autre application. Le résultat de l'interrogation ne contient aucune donnée calculée ou dérivée, c'est-à-dire obtenue à partir d'autres données.

– L'INT ne met à jour aucun GDI. Sur le résultat de l'interrogation figurent un certain nombre de champs, qui sont des données élémentaires (DE). On compte une seule DE pour un champ répétitif. Si deux interrogations ont la même logique de traitement et les mêmes DE (par exemple, résultat papier et résultat écran), on ne comptera qu'une seule INT.

3-5) SOR (Sortie) :

- C'est une fonction élémentaire, significative pour l'utilisateur, relative à l'aspect dynamique du Système d'Information.

- Elle envoie des données vers l'extérieur du domaine et n'effectue aucune mise à jour à l'intérieur de ce domaine. Elle laisse l'application dans un état de cohérence fonctionnelle.
- Par simplification, une sortie correspond à la génération d'un état élémentaire, d'un écran de visualisation ou d'un message à destination D'une autre application, avec des données calculées ou dérivées, c'est à dire obtenues à partir d'autres données.
- Sur chaque sortie figurent un certain nombre de champs, qui sont des données élémentaires (DE). On compte une seule DE pour un champ répétitif.

- Si deux sorties ont la même logique de traitement et les mêmes DE (par exemple, sortie papier et sortie écran), on ne comptera qu'une seule SOR.

Niveau de complexités :

- **Données Elémentaires (DE)**

Chaque GDI ou GDE est composé de données élémentaires. Une DE équivaut à un champ de données. On compte un seul DE par champ répétitif dans les entrées, les sorties, et les interrogations.

• **Sous-ensemble logique de données (SLD)**

Relatif aux GDE et aux GDI. D'un point de vue fonctionnel, ce sont les groupements logiques de GDI ou de GDE qui sont traités simultanément dans l'application.

• **Groupe de données référencées (GDR)**

Relatif aux ENT, SOR ou INT. D'un point de vue fonctionnel, ce sont les groupements logiques de GDI ou de GDE qui sont mis à jour, ou consultés simultanément par les différentes ENT, SOR ou INT.

Avantages : Résident principalement dans le fait qu'il est possible d'effectuer une évaluation grâce à cette méthode très en amont dans le projet – en même temps que les spécifications fonctionnelles [7].

4) Modèles discrètes

Ces modèles ont une forme tabulaire, qui concerne généralement l'effort, la durée, la difficulté et autres facteurs de coût. Ils ont acquis une certaine popularité dans les premiers jours de l'estimation des coûts, et aussi ils sont faciles à utiliser [1].

5) Autres modèles

De nombreux autres modèles existent et les suivants ont été utilisés avec succès dans la pratique [1].

Prix-S : est un modèle d'estimation des coûts développé et maintenu par RCA, New Jersey à partir d'une estimation de projet de taille, le type et la difficulté, l'ordinateur modèle, coût du projet et le calendrier.

SoftCost : concerne la taille, l'effort et la durée au risque d'adresse en utilisant une forme de la probabilité de Rayleigh. Il contient des heuristiques pour guider les estimateurs dans le traitement des nouvelles technologies et les relations complexes entre les paramètres impliqués.

6) Limitations des modèles algorithmiques

Les modèles paramétriques, bien qu'ils ont été les premiers développés et utilisés, souffrent encore de plusieurs limitations [5] :

• Ils se basent essentiellement sur la taille du logiciel en tant que facteur le plus pertinent et négligent souvent l'influence des autres facteurs.

• Ils fixent une forme pour la relation exprimant l'effort en fonction des facteurs de l'effort. Cependant dans plusieurs cas, cette relation ne semble avoir aucune forme prédéterminée.

• Ils nécessitent une calibration quand on veut les appliquer dans des environnements différents de ceux où ils ont été développés.

• Les données disponibles au moment d'estimation sont imprécises et/ou incertaines.

IV) Les modèles non algorithmiques d'estimation de coût

Les modèles non algorithmiques (non-paramétriques) d'estimation des coûts ont été développés pour remédier aux inconvénients cités dans les modèles paramétriques. Ils n'ont pas une équation centrale d'estimation de l'effort. Ainsi, les modèles non-paramétriques n'exigent au préalable aucune forme à la relation exprimant l'effort en fonction des conducteurs du coût. Ils modélisent cette relation en utilisant des techniques d'intelligence artificielle telles que le raisonnement à base de cas, les réseaux de neurones, les systèmes à base de règles et les arbres de décision. Les modèles non-paramétriques représentent donc une alternative prometteuse quand la relation entre l'effort et les conducteurs du coût semblent n'avoir aucune forme prédéfinie [6].

1) Estimation par analogie

Cette méthode se base sur l'existence d'un projet déjà achevé et similaire au projet à développer pour en déduire une estimation de son coût en évaluant les points de similarité et de différence entre l'ancien et le nouveau projet (Le coût du projet est calculé en le comparant à d'autres projets similaires dans le même domaine d'application).Ensuite, l'évaluateur en tenant compte des résultats de cette comparaison génère une estimation du nouveau projet. En effet, cette technique informelle est très souvent utilisée implicitement par l'expert lors de son acte d'estimation [4].

Avantages : Estimations précises si les données concernant des projets similaires sont disponibles.

Inconvénients : Des projets similaires peuvent ne pas exister ; Les données historiques peuvent ne pas être exactes.

2) Jugement de l'expert

Cette technique consiste à consulter un ou plusieurs experts qui utilisent leurs expériences ainsi que leur compréhension du projet afin de fournir une estimation à son coût. Elle était parmi les premières techniques utilisées en estimation des coûts. Il est préférable d'avoir plusieurs valeurs estimées du coût émanant de plusieurs experts pour alléger les problèmes de subjectivité, de pessimisme et d'optimisme (Boehm, 1981). Il y a plusieurs façons de combiner les différentes valeurs estimées du coût d'un projet logiciel. La plus simple est d'utiliser une des méthodes statistiques telles que la moyenne, la médiane ou le mode de toutes les valeurs pour déterminer une estimation du coût. Cette approche présente l'inconvénient d'être sujette aux préjugés des estimations extrêmes. Parmi les techniques utilisées dans cette méthode on peut citer **Delphi**.

La technique **Delphi** fonctionne comme suit [1] :

1) Le coordonnateur présente chaque expert avec un cahier des charges et un formulaire pour les estimations de disques.

2) Chaque expert remplit le formulaire individuel (sans en discuter avec les autres) et est autorisé à poser des questions au coordonnateur.

3) Le coordonnateur prépare un résumé de toutes les estimations des experts (y compris moyenne ou médiane) sur un formulaire de demande d'une autre itération des estimations des experts et la justification pour les estimations.

4) Répétez les étapes 2) -3) autant de tours, le cas échéant.

Avantages : Estimations précises si les experts ont de l'expérience dans des projets similaires ; estimation rapide.

Inconvénients : Estimations non précises s'il y a peu d'expertise.

3) Loi de Parkinson

En utilisant le principe «Le travail se développe pour remplir le volume disponible» de Parkinson [28], le coût est déterminé (non estimé) par les ressources disponibles[1]. Si le logiciel doit être livré en 12 mois et 5 personnes sont disponibles, l'effort est estimé à 60 **mois-personnes**. Bien qu'il donne parfois une bonne estimation, cette méthode n'est pas recommandée car elle peut fournir des estimations très irréalistes.

Avantages : pas d'excès dans les dépenses.

Inconvénients : projets souvent non achevés.

4) Estimation ascendante et descendante

Dans l'estimation ascendante, le logiciel est décomposé en plusieurs tâches, constituant ainsi une arborescence de toutes les tâches du logiciel. Tout d'abord, on estime l'effort nécessaire pour chaque tâche du plus bas niveau dans l'arborescence ; ensuite, on détermine progressivement l'effort des autres tâches, se retrouvant dans un niveau supérieur dans l'arborescence, en combinant les efforts nécessaires associés aux sous-tâches. En général, l'effort nécessaire d'une tâche sera la somme de ceux de ses sous-tâches. Cependant, dans le cas des logiciels complexes, d'autres techniques plus sophistiquées, telles que celles se basant sur des formules mathématiques typiques ou sur des règles d'induction, peuvent être utilisées afin de mettre en valeur la complexité des interfaces de communication entre les tâches [4].

Avantages : Basée sur une analyse détaillée, la méthode est précise si la conception a été effectuée avec suffisamment de détails.

Inconvénients : Peut négliger les facteurs de coût au niveau du système ; Elle exige plus d'efforts d'estimation par rapport à l'estimation descendante ; Difficile d'effectuer l'estimation précoce dans le cycle de vie.

Dans l'estimation descendante, on évalue une estimation globale de l'effort de logiciel ; ensuite, on répartit cet effort total sur toutes les tâches du logiciel [4].

Avantage : Plus rapide et plus facile que l'estimation ascendante ; Exiger un minimum de détails du projet ; Prendre en compte des coûts globaux comme celui de l'intégration, de la gestion de configuration, de la documentation,…

Inconvénient : Fournit peu de détails pour justifier l'estimation ; Moins précis que d'autres méthodes.

→Dans les deux cas de l'estimation ascendante ou descendante, les valeurs estimées de l'effort sont déterminées en utilisant la technique du jugement de l'expert, l'estimation par analogie ou un modèle d'estimation.

V) Conclusion

Aujourd'hui, presque aucun modèle ne peut estimer le coût du logiciel avec un haut degré de précision. Cet état de la pratique est créé parce que : [1]

1) il y a un grand nombre de facteurs interdépendants qui influent sur le processus du développement de logiciels par une équipe de développement donnée, et un grand nombre **d'attributs** de projets (tels que le nombre des écrans d'utilisateurs, la volatilité des exigences du système et l'utilisation des composants de logiciels réutilisables).

2) l'environnement de développement évolue en permanence.

3) le manque de mesure qui reflète bien la complexité d'un système logiciel.

Chapitre 2 :

Les métriques logicielles

I) Introduction

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. La mesure des modèles 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 [18].

II) Les métriques

Les métriques peuvent être classées en trois catégories : [10]

• celles mesurant le processus de développement ;

• celles mesurant des ressources ;

• et celles de l'évaluation du produit logiciel.

Les métriques du processus de développement : 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 des ressources : mesurent les ressources nécessaires à chacune 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 l'évaluation du produit logiciel : détiennent les entités logicielles qui résultent du processus d'activité.

Dans cette partie on s'intéresse aux métriques **d'évaluation du produit logiciel :**

Les métriques d'**évaluation du logiciel** mesurent les qualités du logiciel. Parmi ces métriques, on distingue les **métriques traditionnelles** et les métriques **orientées objet**.

1) Les métriques traditionnelles

Les métriques traditionnelles se divisent en deux groupes : les métriques mesurant **la taille et la complexité**, et les métriques mesurant **la structure du logiciel**. Les métriques mesurant taille et complexité les plus connus sont les métriques de ***ligne de code*** ainsi que les métriques de ***Halstead***. Les métriques mesurant la structure d'un logiciel comme la complexité cyclomatique de ***McCabe*** se basent sur des organigrammes de traitement ou des structures de classe [10].

Le nombre cyclomatique de Mc Cabe : v(G)

La complexité Cyclomatique (complexité de *McCabe*), introduite par Thomas *McCabe* en 1976, est le calcul le plus largement répandu des métriques statiques. Conçue dans le but d'être indépendante du langage, la métrique de *McCabe* indique le nombre de chemins linéaires indépendants dans un module de programme et représente finalement la complexité des flux de donnés.

Il 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.

Pour un programme qui consiste en seulement des états séquentiels, la valeur pour v(G) est 1.

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.

Du fait que le nombre cyclomatique décrive la complexité du flux de contrôle, il est évident que les modules et les fonctions ayant un nombre cyclomatique élevé auront besoin de plus de cas de tests que celles avec une complexité de *McCabe* plus bas. Le principe de base est que chaque fonction devrait avoir un nombre de cas de tests au moins égal au nombre cyclomatique, pour que tous les chemins soient couverts au moins une fois.

Les constructions de langages suivants incrémentent le nombre cyclomatique :

if (...), for(...), while (...), case ...:, catch (...), &&, ||, ?, #if, #ifdef, #ifndef, #elif.

Le calcul commence toujours avec la valeur 1. A ceci on ajoute le nombre de nouvelles branches.

Une fonction devrait avoir une nombre cyclomatique inférieur à 15. Si une fonction a plus que 15 chemins d'exécution il est difficile à comprendre et à tester. Pour un fichier le nombre cyclomatique ne devrait pas dépasser 100.

Les Métriques de Halstead

Les métriques de complexité de *Halstead* qui procurent une mesure quantitative de complexité ont été introduites par l'américain Maurice Halstead. Ils sont basés sur l'interprétation du code comme une séquence de marqueurs, classifiés comme un opérateur ou un opérande.

Toutes les métriques de *Halstead* sont dérivées du nombre d'opérateurs et d'opérandes :

- nombre total des opérateurs uniques (n1)
- nombre total des opérateurs (N1)
- nombre total des opérandes uniques (n2)
- nombre total des opérandes (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

A partir de là, on obtient le Volume du Programme (V) en multipliant la taille du vocabulaire par le logarithme 2 : V = N * log2(n)

Le volume d'une fonction devrait être compris entre 20 et 1000. Le volume d'une fonction, d'une ligne et sans paramètre, qui n'est pas vide est d'environ 20. Une fonction avec un volume supérieur à 1000 comporte probablement trop de choses. Le volume d'un fichier devrait être au minimum à 100 et au maximum à 8000.

Le Niveau de difficulté (D) ou propension d'erreurs du programme est proportionnel au nombre d'opérateurs unique (n1) dans le programme et dépend également du nombre total d'opérandes (N2) et du nombre d'opérandes uniques (n2). Si les mêmes opérandes 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 obtenue par la formule suivante :

E = V * D

Halstead à découvert que diviser l'effort par 18 donne une approximation pour le Temps pour implémenter (T) un programme en secondes.

T = E / 18

Il est même possible d'obtenir le « nombre de bugs fournis » (B) qui est une estimation du nombre d'erreurs dans le programme. Cette valeur donne une indication pour le nombre d'erreurs qui devrait être trouvé lors

du test de logiciel. Le « nombre de bugs fournis » est calculé selon la formule suivante :

B = (E (2/3)) / 3000

Des expériences ont montré qu'un fichier source écrit en C ou C++ contient malheureusement très souvent plus d'erreurs que B ne suggère.

2) Les métriques orientées objets du diagramme de classe

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

Les métriques orientés objets : Prennent en considération les relations entre éléments de programme (classes, méthodes).

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

Métriques	Définition
WMC - WeightedMethods per Class (poids des méthodes par classe)	la somme de la complexité de ses méthodes.
DIT - Depth of InheritanceTree (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 -	C'est une métrique qui représente

CouplingBetween Object classes (couplage entre objets d'une classe)	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 cohesion 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.

Tableau 1 : Métriques diagramme de classe

3) Les métriques des points de fonction et lignes de code

La mesure des points de fonction est la méthode principale pour mesurer la fonction inhérente du produit logiciel. Elle est faite à partir des documents de spécification produits dans les phases préliminaires du développement [9]. Durant la construction du logiciel, la métrique du nombre de points de fonction identifie et exprime la taille fonctionnelle du système.

Premièrement, pendant l'implémentation du système logiciel, le nombre de points de fonction peut être utilisé afin d'exprimer la productivité qui est représentée par le nombre de points de fonction achevé par heure de travail. Également, cette métrique est souvent utilisée pour calculer la densité des défauts en divisant le nombre de défauts par le nombre de points de fonction. D'un autre côté, on peut suivre la progression ou la complétion de la construction par examen du nombre de points de fonction spécifiés, conçus, implémentés et testés, qui sont identifiés dans les différentes activités de la construction.

Deuxièmement, les points de fonction peuvent être **obtenus dans les phases initiales du développement** comme la spécification ou la conception du système.

La méthode des points de fonction a été initialement formulée par Albrecht en 1979. Cependant, la spécification du système à implémenter sera découpée en plusieurs composants selon des types prédéfinis.

Les métriques des Lignes 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 [10].

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).

→Analyse de la mesure des points de fonction dans l'estimation logicielle : [11]

Malgré les facteurs subjectifs de la mesure introduits par les jugements subjectifs des différents estimateurs dans le comptage du nombre brut des points de fonction et l'évaluation des facteurs de complexité, le bénéfice majeur des métriques de points de fonction réside dans le fait qu'elle prévoit et produit la taille logicielle estimée en nombre de points de fonction dans **les phases initiales du développement** d'une manière plus rigoureuse que **le nombre de lignes de code**.

Le tableau ci-dessous montre les avantages de la métrique des **points de fonction** par rapport à la métrique du nombre de **lignes de code** dans l'estimation logicielle :

La méthode des points de fonction	Le nombre de lignes de code
• Indépendante de langages, des outils ou des méthodologies de programmations utilisés. • Estime la taille fonctionnelle à partir des fichiers de spécification des besoin du clients dans les phases initial de développement, facile à adapter aux changements des besoins et ainsi de refaire la prédiction de la taille. • Estimation basée sur la vue externe de l'utilisateur représentant les fonctions du système.	• Forte variabilité sur le nombre de lignes de code des projets construits pour différents langages de programmation ; il est impossible de comparer la taille en ligne de code sur des projets développé en différents langages. • Manque d'une définition uniforme sur le sens d'une ligne de code. Il est donc très difficile de faire des études comparatives en utilisant le nombre de lignes de code comme une mesure de la taille logicielle. • Très difficile de prévoir la taille en nombre de lignes de code précisément en fonction des spécifications ou de la conception du système à implanter.

Tableau 2 : Comparaison entre PF et LOC

III) Conclusion

Dans ce chapitre nous avons présenté quelques métriques concernant les points de fonction, ligne de code ainsi que d'autres comme les métriques du diagramme de classe, qui peuvent être calculées dans les phases initiales du développement d'un projet logiciel.

Chapitre 3 :

Méthodes datamining

I) Introduction

Le datamining est un processus de découverte de règles, relations, corrélations et/ou dépendances à travers une grande quantité de données, grâce à des méthodes statistiques, mathématiques et de reconnaissances de formes.

Dans ce chapitre nous allons présenter quelques méthodes qui vont être utilisées par la suite pour estimer l'effort d'un nouveau projet logiciel.

II) Réseaux de neurones

1) Définition

Les Réseaux de neurones sont des réseaux fortement connectés de processeurs élémentaires fonctionnant en parallèle.

Chaque processeur élémentaire calcule une sortie unique sur la base des informations qu'il reçoit. [12]

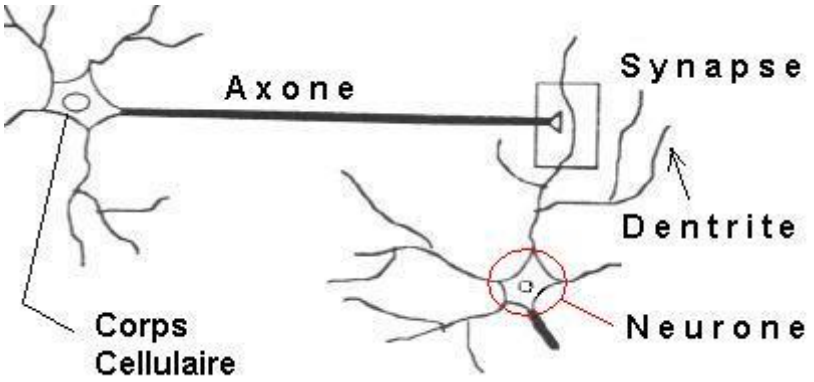

Figure 1 : Neurone biologique

Structure d'un neurone artificiel :

En équivalence avec le système biologique, Ce processeur est appelé neurone.

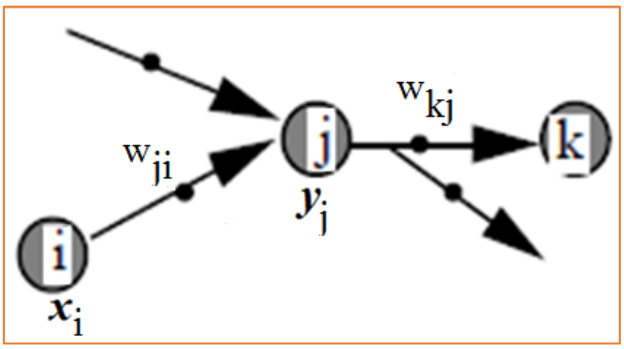

Figure 2 : Neurone formel

Pour le neurone d'indice *j* :

- les entrées sur celui-ci sont de poids *wji.*
- les connexions avals sont de poids *wkj*.

On distingue deux phases :

1. Calcul de la somme pondérée des entrées :
2. Une fonction de transfert calcule la valeur de l'état du neurone (sortie) $y_j = g(a_j)$

C'est cette valeur qui sera transmise aux neurones avals.

La fonction de transfert :

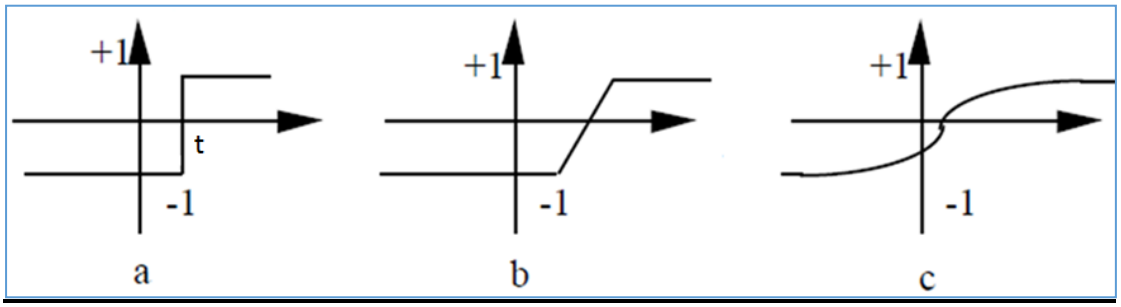

Figure 3 : les fonctions de transfert

a : fonction à seuil (t : la valeur du seuil)

b: linéaire par morceaux

c : sigmoïde

2) Le Perceptron

Le perceptron 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 [12].

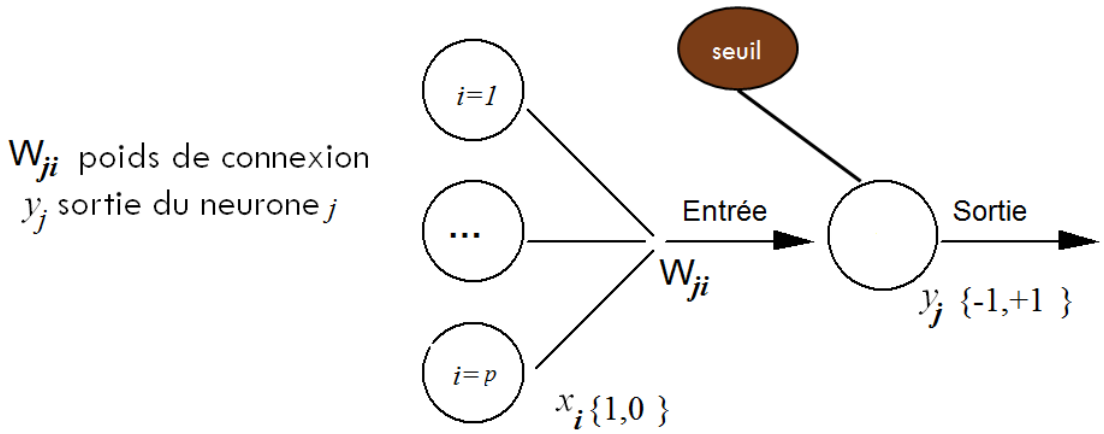

Figure 4 : Perceptron

Un réseau de neurones monocouche est caractérisé par :

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

A noter : [12]

$\boldsymbol{X} = (\boldsymbol{x_i})_{1 <= i <= p}$: les p **informations d'entrée**

$\boldsymbol{w_{ji}}$, 1 <= *i*<= p et 1 <= *j* <= q : les **poids de connexion**

$\boldsymbol{y_j}$: la **sortie** du *j*-ème neurone

$\boldsymbol{a_j}$: la donnée d'entrée (**somme pondérée**) du *j*-ème neurone.

Seuil *:* **seuil d'activation** du neurone

On parle aussi du coefficient de biais wj0, pour **ajuster** la sensibilité du neurone On a donc l'équation suivante :

$1 \leq j \leq q$

$$y_j = g(a_j) = g(\sum_{i=0}^{p}(w_{ji}) * x_i$$

Généralement, le neurone est activé si l'activation on atteint le seuil de la fonction d'activation lorsque la somme pondérée des informations d'entrée vaut le coefficient de biais (wj0).

L'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.

3) Le perceptron multicouche

Le perceptron multicouche est un classificateur linéaire de type réseau neuronal formel organisé en plusieurs couches au sein desquelles une information circule de la couche d'entrée vers la couche de sortie uniquement. Chaque couche est constituée d'un nombre variable de neurones, les neurones de la couche de sortie correspondant toujours aux sorties du système. [13]

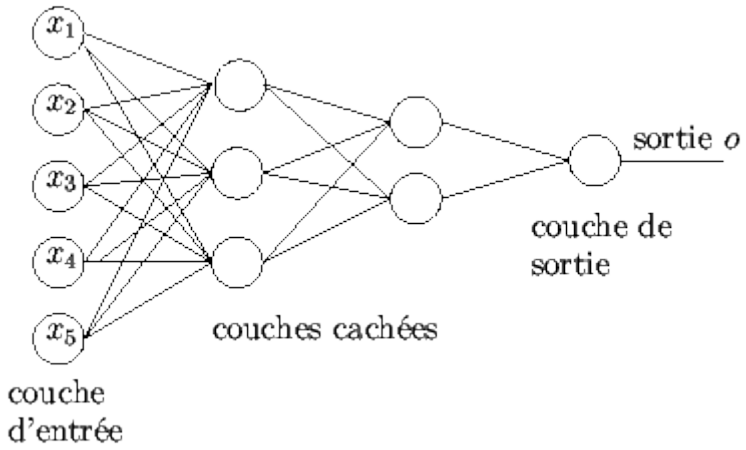

Figure 5 : Perceptron multicouche

- Couche d'entrée : x1, …. Xn
- Couches cachées
- Couche de sortie

III) Régression linéaire

1) Définition

un modèle de **régression linéaire** est un modèle de régression d'une variable expliquée sur une ou plusieurs variables explicatives dans lequel on fait l'hypothèse que la fonction qui relie les variables explicatives à la variable expliquée est linéaire dans ses paramètres. On parle aussi de **modèle linéaire** ou de **modèle de régression linéaire**.

En général, le modèle de régression linéaire désigne un modèle dans lequel l'espérance conditionnelle de **y** sachant **x** est une transformation affine de **x**. Cependant, on peut aussi considérer des modèles dans lesquels c'est la médiane conditionnelle de **y** sachant **x** ou n'importe quel quantile de la distribution de **y** sachant **x** qui est une transformation affine de **x** [14].

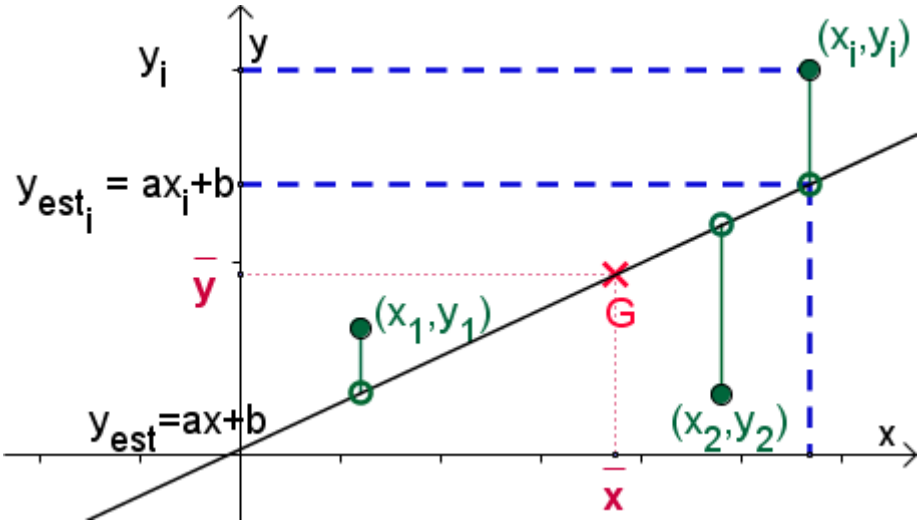

Figure 6 : droite de la régression linéaire

2) Modèle linéaire

Un modèle de régression linéaire simple est de la forme : [15]

$$Y = \beta 0 + \beta 1 X + \varepsilon$$

Où

- Y est la variable dépendante.
- β0 et β1 sont les coefficients (ordonnée à l'origine et pente).
- X est la variable indépendante (variable explicative).
- ε est une erreur aléatoire.

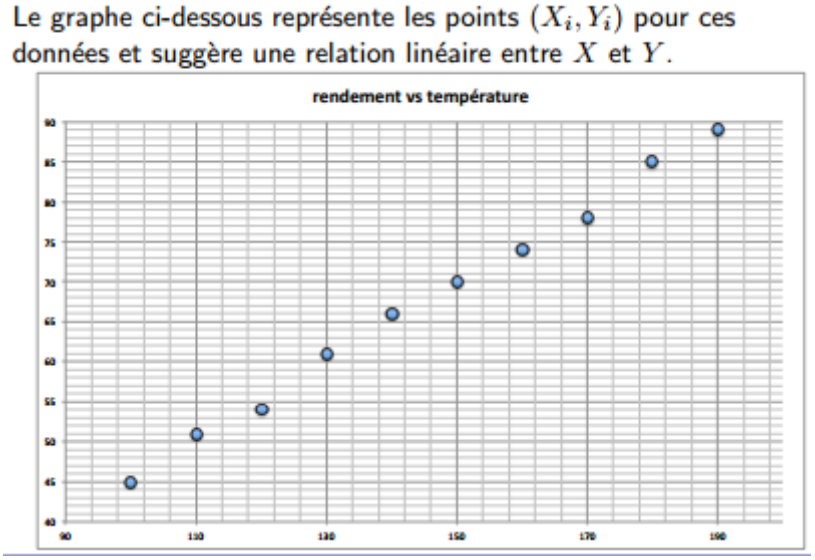

Figure 7 : exemple régression linéaire

IV) SVM

.

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.

Un SVM, comme un perceptron, trouve un séparateur linéaire entre les points de données de deux classes différentes.

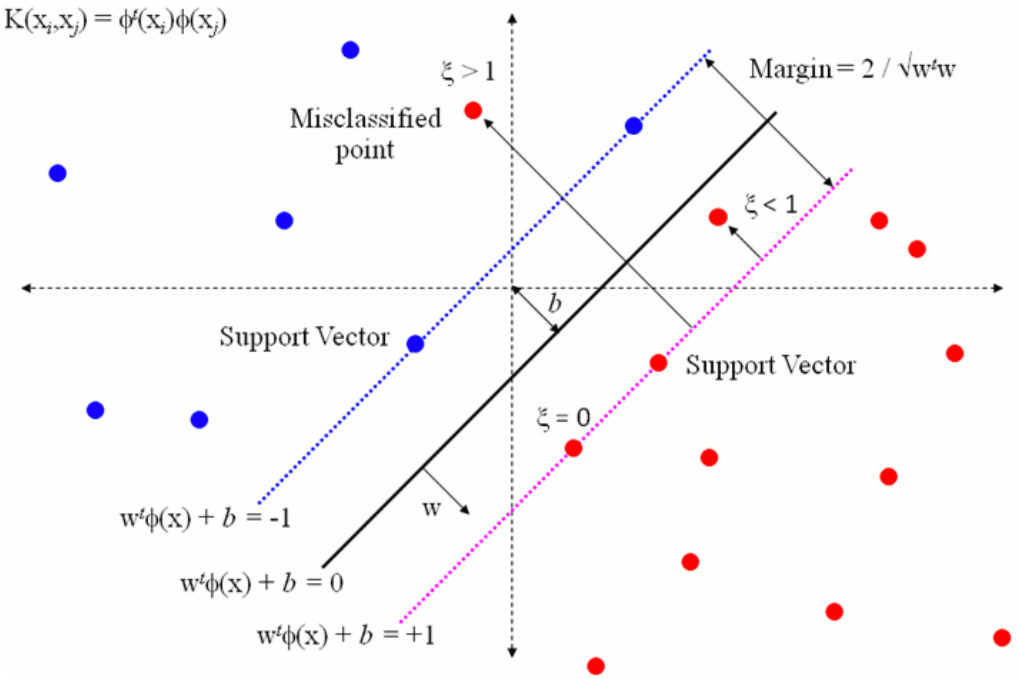

Figure 8 : SVM

2) 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 paramètre 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) Les 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).

V) Arbre de décision

1) Définition

Un arbre de décision est un outil d'aide à la décision représentant un ensemble de choix sous la forme graphique d'un arbre. Les différentes décisions possibles sont situées aux extrémités des branches (les « feuilles » de l'arbre), et sont atteints en fonction de décisions prises à chaque étape. L'arbre de décision est un outil utilisé dans des domaines variés tels que la sécurité, la fouille de données, la médecine, etc. Il a l'avantage d'être lisible et rapide à exécuter. Il s'agit de plus d'une représentation calculable automatiquement par des algorithmes d'apprentissage supervisé.

2) Exemples

-Décider si un patient est malade ou bien portant selon sa température et s'il a la gorge irritée : [16]

→Arbre de décision :

➔ 2 classes (malade, bien portant)
➔ 2 variables (température, gorge irritée)

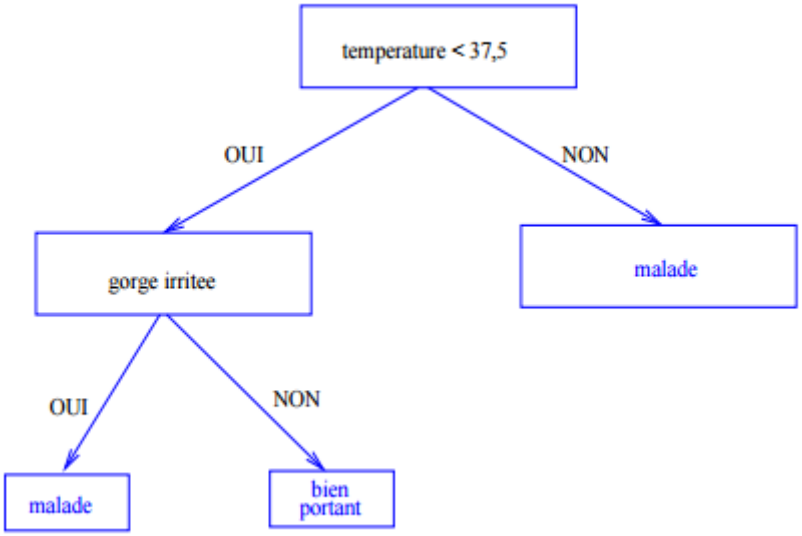

Figure 9 : exemple arbre de décision

-Construire un arbre de décision qui classe et détermine les caractéristiques des clients qui consultent leurs comptes sur internet

Variables

M : moyenne des montants sur le compte

A : âge du client

R : lieu de résidence du client

E : le client à des études supérieures ?

I : le client consulte ses comptes sur internet ?

Client	M	A	R	E	I
1	moyen	moyen	village	oui	oui
2	élevé	moyen	bourg	non	non
3	faible	âgé	bourg	non	non
4	faible	moyen	bourg	oui	oui
5	moyen	jeune	ville	oui	oui
6	élevé	âgé	ville	oui	non
7	moyen	âgé	ville	oui	non
8	faible	moyen	village	non	non

Tableau 3 : les clients (ex arbre de décision)

Construction selon la variable Montant (M) :

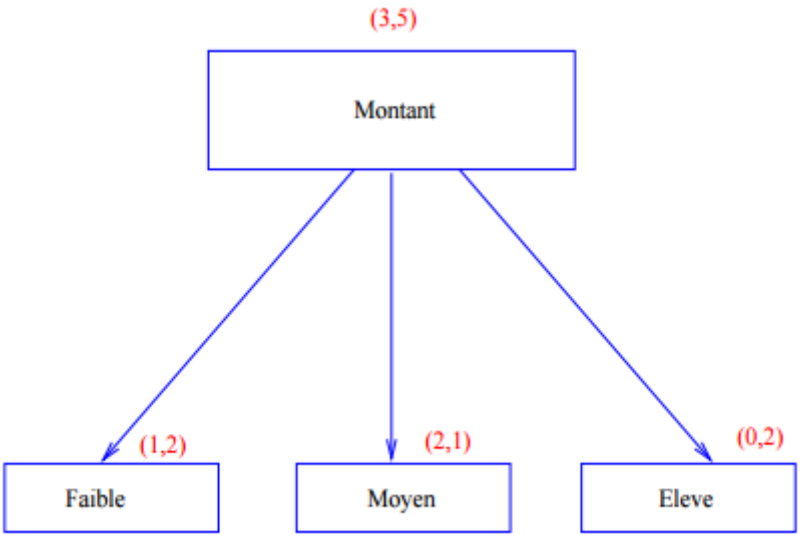

Figure 10 : Variables montant

Chapitre 4 :

Application réalisée et tests

I) Introduction

Pour la réalisation de l'application de la prédiction du coût d'un projet logiciel, nous allons passer par le choix d'une base de données contenant l'ensemble des métriques les plus pertinentes.

Afin d'atteindre notre objectif, nous avons appliqué quelques méthodes de classification Datamining pour estimer l'effort du projet, puis, on a essayé d'évaluer la précision de chacune de ces méthodes en utilisant les outils de développements citer dans les paragraphes suivants.

II) Outils de développements

1) Weka

Est une suite de logiciels d'apprentissage automatique et d'exploration de données écrites en Java, Développé à l'université de Waikato en Nouvelle-Zélande.

Son but est de :

- ✓ Réaliser des tâches de fouilles de données.
- ✓ Appliquer des algorithmes d'apprentissage.
- ✓ Analyser les résultats et les performances d'un classificateur.

La version utilisée :

La version utilisée est 3.8.0.

Figure 11 : Outil weka

Les composants de weka :

Weka possède plusieurs Composants : [17]

- Outils de prétraitement des données (explorer).
  - Sélection, transformation, combinaison d'attributs.
  - Normalisation.
  - Ré échantillonnage, ...

- Algorithmes pour l'exploration de données.
  - Classification non supervisée.
  - Classification supervisée.

- Analyse de résultats.
- Comparaison d'algorithmes.
- Plusieurs interfaces graphiques.

2) NetBeans

NetBeans est un environnement de développement intégré (EDI), placé en open source par Sun en juin 2000 sous licence CDDL (Common Development and Distribution License) et GPLv2. 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 (comme Python ou Ruby) par l'ajout de greffons. Il comprend toutes les caractéristiques d'un IDE moderne (éditeur en couleur, projets multi-langage, refactoring, éditeur graphique d'interfaces et de pages Web). [18]

3) Java Eclipse

Eclipse IDE est un environnement de développement intégré libre (le terme Eclipse désigne également le projet correspondant, lancé par IBM) extensible, universel et polyvalent, permettant potentiellement de créer des projets de développement mettant en œuvre n'importe quel langage de programmation. Eclipse IDE est principalement écrit en Java (à l'aide de la bibliothèque graphique SWT, d'IBM), et ce langage, grâce à des bibliothèques spécifiques, est également utilisé pour écrire des extensions.

III) Présentation de l'application

1) Choix de la base de donnée

Pour le choix de la base de données, on va sélectionner celle qui contient les métriques de **points de fonction** les plus pertinentes pour estimer l'effort d'un nouveau projet.

→La raison pour laquelle on a choisi les points de fonction c'est qu'ils peuvent être obtenus dans les phases initiales du développement.

Nous présentons ci-dessous l'ensemble des bases de données contenant l'historique des projets décrits par les **points de fonction** :

Base de données	**Année**	**Attributs**
Albrecht	2009	Input numeric Output numeric Inquiry numeric File numeric FPAdj numeric RawFPcounts numeric AdjFP numeric Effort numeric
Desharnais	1989	Project numeric TeamExp numeric ManagerExp numeric YearEnd numeric Length numeric Effort numeric Transactions numeric Entities numeric PointsNonAdjust numeric Adjustment numeric PointsAjust numeric Language {1,2,3}
MAXWELL	-	size_D numeric duration_D numeric app {InfServ,TransPro,CustServ,ProdCont,MIS} har {PC,Mainfrm,Multi,Mini,Network}

		dba {Relatnl,Sequential,None,Other} ifc {GUI,TextUI} source {Outsrced,Inhouse} nlan {1,3,2,4} telonuse {No,Yes} t01 {1,3,2,4,5} t02 {1,3,2,4,5} t03 {1,3,2,4,5} t04 {1,3,2,4,5} t05 {1,3,2,4,5} t06 {1,3,2,4,5} t07 {1,3,2,4,5} t08 {1,3,2,4,5} t09 {1,3,2,4,5} t10 {1,3,2,4,5} t11 {1,3,2,4,5} t12 {1,3,2,4,5} t13 {1,3,2,4,5} t14 {1,3,2,4,5} t15 {1,3,2,4,5} effort_D numeric
China	-	ID numeric AFP numeric Input numeric Output numeric Enquiry numeric File numeric Interface numeric Added numeric Changed numeric Deleted numeric PDR_AFP numeric PDR_UFP numeric NPDR_AFP numeric NPDU_UFP numeric Resource numeric Dev.Type numeric Duration numeric N_effort numeric Effort numeric
KEMERER	-	ID numeric Language numeric

		Hardware numeric Duration numeric KSLOC numeric AdjFP numeric RAWFP numeric EffortMM numeric
Software effort estimation at feature level	2007	ID→Id du projet Effort→En homme-heure IntComplx→Complexité du calcul interne (1-très faible, 2-faible, 3-moyen, 4-élevé, 5-très élevé) DataFile→Nombre de fichiers de données DataEn→ Nombre de données d'entrées DataOut→ Nombre de données de sorties UFP→Compter le nombre des points de fonction non ajusté Lang→language utilisée (C++, Java, VB, Java Script, VB Script, SQL, Php, Perl, Asp, Html, XML, Others) Tools→ outils de développement (VJ++, VB, Delphi, VisualCafe, JUnit, PowerBuilder, BorlandC++, Others) ToolExpr→Niveau d'expérience concernant l'outil de développement (Gamme de nombre de mois d'expérience, par exemple [2, 5] pour 2 à 5 mois, le niveau d'expérience minimum dans l'equipe est de 2 et 5 est le maximum dans l'équipe) AppExpr→niveau d'experience d'application (1-très faible, 2-faible, 3-moyen, 4-élevé, 5-très élevé) TeamSize→ Taille de l'équipe pour la mise en œuvre de l'objet DBMS→ Systèmes de base de données (Oracle, Access, SQLServer, MySQL, Others) Method→ Méthodologie (OO, SA, SD, RAD, JAD, MVC ….) AppType→ Type d'Architecture système / application (B/S, C/S, BC/S, Centered, Other)
COSMIC	2009	PID Num_FP_Nesma Num_ILF Num_EIF Num_EI

		Num_EO Num_EQ Num_CFP Num_FuncProc

Tableau 4 : Bases de données PF

Pour notre cas, la base de données la plus pertinente pour estimer l'effort du projet logiciel est : **Software effort estimation at feature level**

2) Application et Résultats

La fenêtre principale :

Figure 12 : Menu Principale

- L'application contient 2 sections :
  1) Estimation du coût d'un nouveau projet logiciel en utilisant les différentes techniques de classification datamining.

2) Précision de l'évaluation de chacune des méthodes utilisées afin de comparer entre ces dernières.

- Il est similaire au fonctionnement de l'outil weka.
- Les API des algorithmes sont importées depuis le toolkits Weka.

Le Format d'entré :

- Le format d'entrée pour notre application est l'ARFF (Attribute Relation File Format).

Caractéristiques du format de fichier ARFF :

- Les commentaires sont précédés de %
- La base de donnée est représenter par : @RELATION [Nom de la relation]
- Les attributs (les métriques) sont représentés par : @ATTRIBUTE [Nom de l'attribut] [type]
  - Pour le type il peut être soit :
    - Numeric.
    - String.
    - Nominal {val1,…, valn}.
    - date [<date-format>].
- Les données : @DATA

  Val1, Val2,…, Valn

  Val1, Val2,…, Valn

  ……

L'estimation du coût :

- Pour faire l'estimation on a besoin de 3 fichiers :
  - 1 fichier pour l'apprentissage.
  - 1 fichier pour le test.
  - 1 fichier pour stocker le résultat du test.

Cette interface permette de choisir la méthode à appliquer :

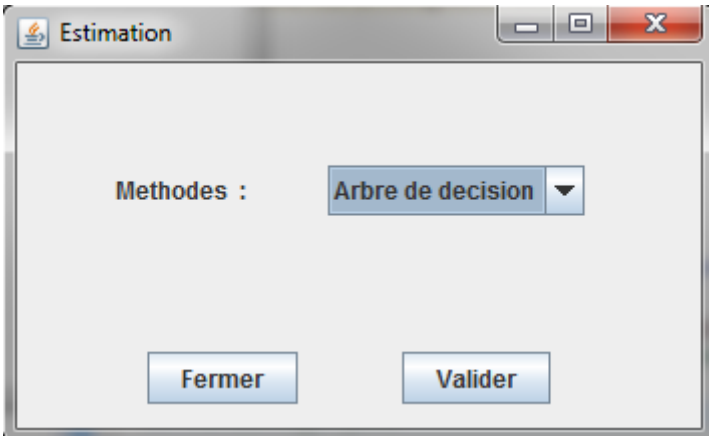

Figure 13 : Estimation choix de la méthode

Dans cet exemple on a choisi comme méthode l'arbre de décision. Après le choix de la méthode on passe à importer le fichier d'apprentissage :

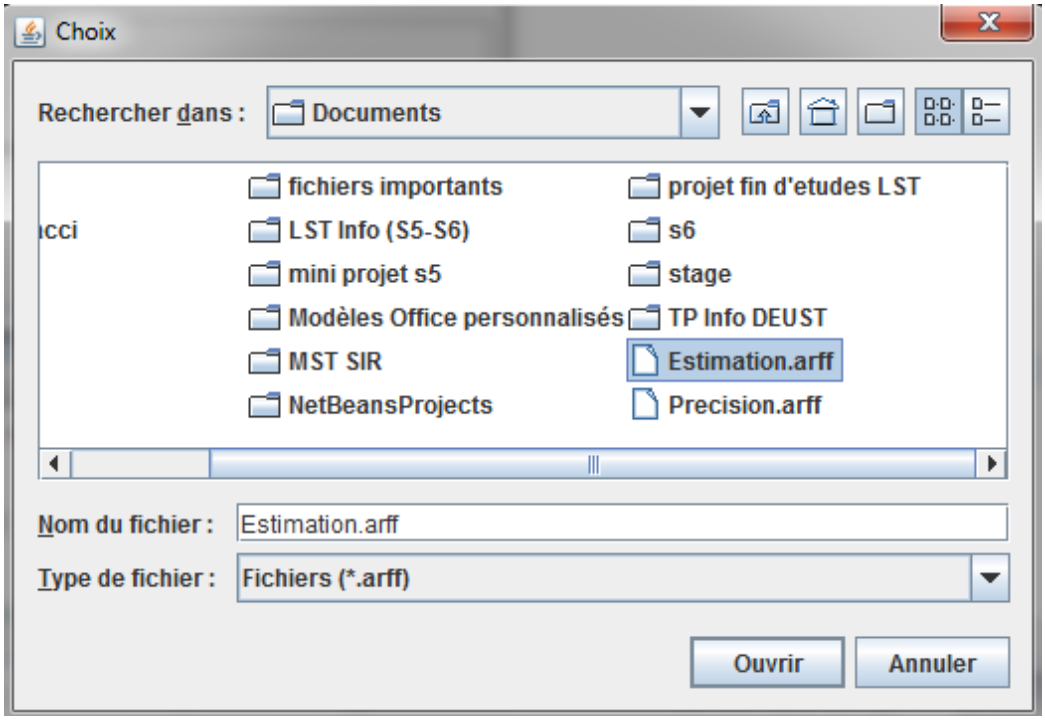

Figure 14 : Choix de la base de données

Fichier d'apprentissage :

```
@data
%236,1,1,2,1,1,0,'Php,_Html,_Sql,_JavaScript','Vim,_Emacs,_Microsof
%101,2.5,5,18,94,?,?,sql,mySQLweb,'[2,60]',4,[1],mysql,?,BC/S
%310,10,2,2,1,1,36,'C#,_ASP.Net_SQL','Visual_Studio.Net_2003,_Micrc
%308,20,3,1,2,4,66,'C#,_ASP.Net_SQL','Visual_Studio.Net_2003,_Micrc
%306,40,3,3,2,1,50,'C#,_ASP.Net_SQL','Visual_Studio.Net_2003,_Micrc
```

Fichier de teste :

→L'attribut de l'effort est le 2ème :

```
@data

236,?,1,2,1,1,0,'Php,_Html,_Sql,_JavaScript','Vim,_Emacs,_Mi
101,?,5,18,94,?,?,sql,mySQLweb,'[2,60]',4,[1],mysql,?,BC/S
310,?,2,2,1,1,36,'C#,_ASP.Net_SQL','Visual_Studio.Net_2003,_
308,?,3,1,2,4,66,'C#,_ASP.Net_SQL','Visual_Studio.Net_2003,_
306,?,3,3,2,1,50,'C#,_ASP.Net_SQL','Visual_Studio.Net_2003,_
```

Résultats :

```
@data
236,1,1,2,1,1,0,'Php,_Html,_Sql,_JavaScript','Vim,_Emacs,_Micro
101,3.875,5,18,94,?,?,sql,mySQLweb,'[2,60]',4,[1],mysql,?,BC/S
310,19.04,2,2,1,1,36,'C#,_ASP.Net_SQL','Visual_Studio.Net_2003,
308,30,3,1,2,4,66,'C#,_ASP.Net_SQL','Visual_Studio.Net_2003,_Mi
306,30,3,3,2,1,50,'C#,_ASP.Net_SQL','Visual_Studio.Net_2003,_Mi
```

→Après avoir appliqué l'apprentissage, les résultats d'estimations sont stockés dans un fichier, et après avoir fait plusieurs tests en utilisant les méthodes (**SVM, arbre de décision, régression linéaire, perceptron multicouche**) on a trouvé les résultats suivants :

Valeur Originale	Arbre de décision	SVM	Régression linéaire	Perceptron Multicouche
1	1	1.01	4.89	1.24
2.5	3.875	1.95	4.89	3.16
10	19.04	29.32	4.89	34.39
20	30	20.02	4.89	22.51
40	30	39.89	4.89	42.44

Tableau 5 : Comparer les efforts

→ D'après ces résultats on constate que la régression linéaire ne donne pas de résultat satisfaisant, par contre **SVM** montre son efficacité par rapport aux autres.

Précision d'évaluation des méthodes :

En cliquant sur Précision, on choisit la méthode par exemple arbre de décision.

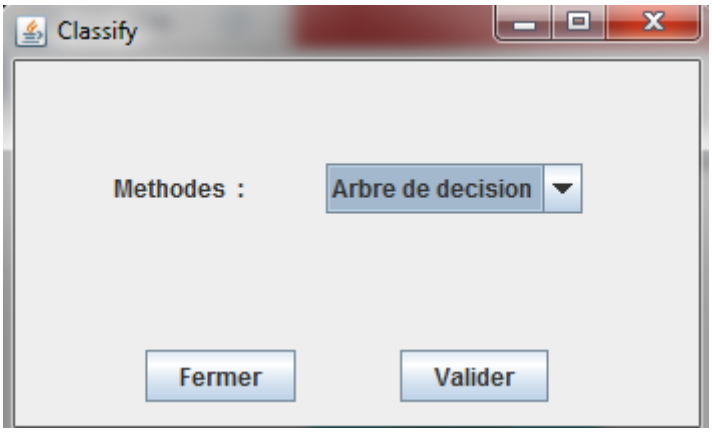

Figure 15 : Précision choix de la méthode

Cette fenêtre permet de choisir l'interface :

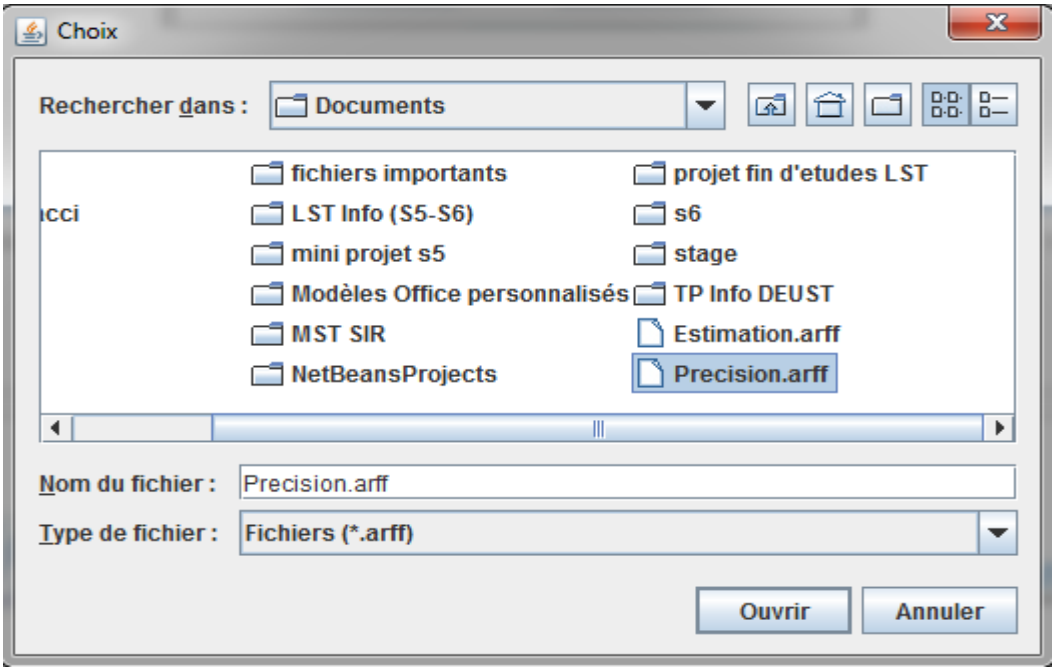

Figure 16 : Choix de la base de données

On obtient les informations suivantes :

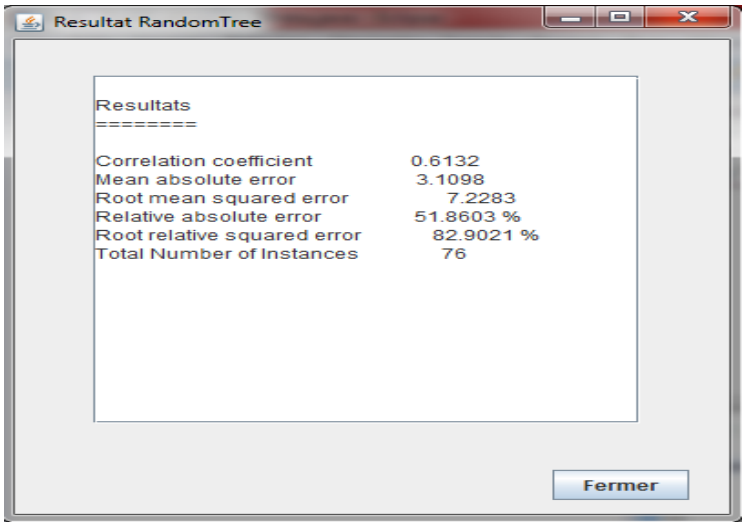

Figure 17 : Résultats de Précision

Corrélation coefficient :[19]

Le coefficient de corrélation de Bravais-Pearson est un indice statistique qui exprime l'intensité et le sens (positif ou négatif) de la relation linéaire entre deux variables quantitatives. C'est une mesure de la liaison linéaire, c'est à dire de la capacité de prédire une variable x par une autre y à l'aide d'un modèle linéaire.

- une valeur proche de +1 montre une forte liaison entre les deux caractères. La relation linéaire est ici croissante (c'est-à-dire que les variables varient dans le même sens).
- une valeur proche de -1 montre également une forte liaison mais la relation linéaire entre les deux caractères est décroissante (les variables varient dans le sens contraire).
- une valeur proche de 0 montre une absence de relation linéaire entre les deux caractères.

Mean absolute error : [20]

L'erreur absolue moyenne est une quantité utilisée pour mesurer la façon dont les résultats des prédictions sont proches.

Root mean squared error : [21]

Le carré moyen des erreurs représente l'écart type des différences entre les valeurs prédites et les valeurs observées.

→Après avoir effectué le test sur les quatre méthodes on obtient les résultats suivants :

Précision	Arbre de décision	SVM	Régression linéaire	Perceptron Multicouche
Correlation coefficient	0.6132	0.7504	-0.2644	0.7979
Mean absolute error	3.1098	2.6597	5.9965	2.8173
Root mean squared error	7.2283	6.2764	8.7191	5.6413
Relative absolute error	51.8603%	44.3547%	100%	46.9824%
Root relative error	82.904%	71.9853%	100%	64.7004%

Tableau 6 : Facteurs de précision

→D'après ces résultats on constate que pour la prédiction de l'effort, les méthodes **SVM** et **Perceptron multicouche** donne de bonnes résultats puisqu'ils ont le coefficient de corrélation plus proche à +1 et les erreurs relatives et absolues sont les plus bas par rapport aux autres, par contre la régression linéaire ne donne pas de résultats satisfaisants, puisque son coefficient de corrélation est proche de 0.

IV) Conclusion

Dans ce chapitre nous avons présenté les différentes bases de données basées sur les points de fonction. Parmi ces bases de données, nous avons pris celle qui est la plus pertinente.

On appliquant les méthodes datamining (*Arbre de décision*, *SVM*, *Régression linéaire*, *Perceptron multicouche*) on constate que ceux les plus efficaces aux niveaux de la prédiction sont les **SVM** et **Perceptron Multicouche**.

Conclusion et perspectives

Afin d'atteindre l'objectif de ce travail, nous avons établi une étude approfondie sur les métriques logicielles et plus particulièrement les métriques qui peuvent être obtenues dans les étapes initiales d'un projet logiciel.

Durant ce travail, nous avons trouvé un manque de base de données sur des métriques très pertinentes et très prometteuses. Ce fait nous a poussés à travailler avec les métriques de points de fonction puisque ces dernières peuvent être obtenues dans les phases initiales du développement. Dans la suite, nous avons appliqué des méthodes Datamining (Arbre de décision, SVM, Perceptron multicouches, Régression linéaire) et nous avons trouvé que les deux méthodes ***SVM et Perceptron Multicouches*** montrent leurs efficacités au niveau de la prédiction.

Les résultats obtenus nous ont permis d'envisager d'autres pistes de recherches dans le futur. Nous citons à titre d'exemple l'exploitation des diagrammes de cas d'utilisation (les Uses case) et de classe afin d'extraire, dès le début d'un projet, des métriques très pertinentes permettant de quantifier d'une manière rationnelle et efficace un logiciel sachant que le développement Orienté Objet domine le marché.

Bibliographie

[1] Software Cost Estimation réalisé par : Hareton Leung Zhang Fan ; Department of Computing The Hong Kong Polytechnic University.

[2] SOFT COMPUTING TECHNIQUES FOR SOFTWARE PROJECT EFFORT ESTIMATION réalisé par : Sumeet Kaur Sehra, Yadwinder Singh Brar, and Navdeep Kaur.

[3] Data mining techniques for software effort estimation to improve cost efficiency réalisé par : Ms. K. Gayathiri, Dr. T. Nalini,Dr. V. Khanaa.

[4] Thèse présentée comme exigence partielle du doctorat en informatique cognitive par Ali idri ; titre de thèse : un modèle intelligent d'estimation des couts de développements de logiciels, pp.11-15.

[5] Thèse présentée à l'École Nationale Supérieure d'Informatique et d'Analyse des Systèmes pour obtenir le diplôme de DOCTORAT DISCIPLINE : Sciences Appliquées SPÉCIALITÉ : Informatique Estimation des coûts de développement de logiciels par un réseau neuronal RBF flou par Abdelali ZAKRANI, pp.12-27.

[8] Chapitre 5 : Les méthodes d'estimation MAKKES M. ISET de Charguia (Sem1- 11/12).

[9] Cours L'estimation de cout par Mme Loubna LAMRINI.

[10] Les métriques (McCabe, Halstead), l'index de maintenabilité et leur influence sur la qualité par Klaus LAMBERTZ, Verifysoft Technology.

[11] Les métriques appliquées dans la construction de logiciel mémoire présenté comme exigence partielle de la maîtrise en informatique par Hao Wang mai 2007.

[12] Cours Le réseau de neurones artificiel par Mme Aicha Majda.

[14] Régression linéaire simple MTH2302D S. Le Digabel, Ecole Polytechnique de Montréal H2016.

[16] Arbres de décision Cours d'analyse de donnéesUniversité Paris I.

[17]Cours Weka Brigitte Bigi LPL - Equipe C3I 15 f´evrier 2011.

[18]La mesure des modèles par les modèles : une approche générative ; Martin Monperrus ; 2011, p4.

[19] Cours de statistiques à distance, élaboré par Zarrouk Fayçal, ISSEP Ksar-Said, 2011-2012 1 ÉTUDE DE LA RELATION ENTRE DEUX VARIABLES (le coefficient de corrélation).

Webographie

[6]http://www.memoireonline.com/04/11/4456/m_Un-atelier-de-genie-logiciel-dedie-a-lestimation-du-cout-logiciel2.html

[7] http://users.polytech.unice.fr/~hugues/GL/COCOMO/cocomo.html

[13]https://fr.wikipedia.org/wiki/Perceptron_multicouche

[14]https://fr.wikipedia.org/wiki/R%C3%A9gression_lin%C3%A9aire

[18]https://fr.wikipedia.org/wiki/NetBeans

[20]https://en.wikipedia.org/wiki/Mean_absolute_error

[21]https://en.wikipedia.org/wiki/Root-mean-square_deviation