Université De Montpellier
Faculté Des Sciences

Niveau : Master 2

Module : IA pour le Génie Logiciel

HAI916I

Synthèse : Software Defect Identification using Machine Learning Techniques

---

Supervisé par :
M. Abdelhak-Djamel Seriai

Réalisé par :
ALLOUCH Yanis
KACI Ahmed

2021/2022

Table des matières

1 Introduction

Ce rapport, aura pour objectif de synthétiser un article scientifique dans le domaine du machine learning. Nous avons choisi l'article de recherche intitulé "**Software Defect Identification using Machine Learning Techniques**"[1].

Tout d'abords, nous allons définir le contexte et le problème traiter dans cet article, ainsi que les solutions existantes.

Puis nous détaillerons la solution choisie et la démarche suivie ainsi que les résultats produits.

Enfin, nous donnerons notre avis personnel sur le travail réalisé dans cette recherche et nous terminerons en proposant une autre solution qui pourrait être vue comme une alternative ou perspective.

2 Contexte et problème traiter

Cet article du domaine de l'intelligence artificiel, expose les analyses, expériences et résultats d'une recherche ayant pour objectif de proposer un modèle permettant la détection de bugs dans le code source d'un logiciel le plus tôt possible dans le cycle de vie d'un projet. Et ce en se basant sur les méthodes d'apprentissage automatique pour répondre à un problème crucial dans le domaine du génie logiciel qui consiste en le développement d'un logiciel de qualité qui réponds aux attentes des clients dans des délais et un budget limité.

3 Solutions connues

Les méthodes actuelles d'ingénierie logicielle présentent des stratégies de "prévention des bugs" plutôt que de "détection des bugs". Les méthodes de prévention de bugs sont réalisées grâce à des processus de gestion de la propagation des erreurs. Les révisions et les inspections dans ces processus sont des moyens traditionnels de réduire activement les erreurs issues de la phase de développement. Cependant, elles sont généralement assez coûteuses et entraînent une augmentation considérable des coûts d'un projet.

4 Les motivations derrière la solution proposée

Alternativement, il est proposé dans cette recherche, la mise en œuvre des techniques d'apprentissage automatique permettant la reconnaissance d'erreurs durant le cycle de développement. Ces techniques paraissent bien adaptées à l'identification et à la prédiction des bugs dans les projets logiciels.

5 Principe et grandes étapes de la solution proposée

Dans cette recherche qui consiste à prédire l'ampleur d'un éventuel bug tel que sa gravité, sa priorité, etc. en se basant sur les métriques de code. Il a été conçu un système d'apprentissage automatique ayant comme données en entrée les attributs (code métrique) de chaque module/fonction d'un projet utilisé pour les tests. Comme ces données ont de nombreuses valeurs corrélées, une analyse en composantes principales (ACP) est effectuée pour transformer les variables corrélées en variables non corrélées. Cette élimination de la corrélation permet de réduire la di-

mensionnalité de ces ensembles de données. Le système est ensuite entraîné avec cet ensemble de données optimisé en utilisant les algorithmes basés sur la régression tels que l'arbre de décision (DT), le Perceptron multicouche (MLP) et les fonctions de base radiales (RBF). Les sorties du système indiquent le nombre d'erreurs total par module/fonction.

5.1 Les techniques utilisées

Le modèle proposé dans cet article est un modèle basé sur techniques de régression pour détecter et identifier les défauts logiciels. Ainsi, deux systèmes d'apprentissage ont été conçu comme suit :

— Dans le système d'apprentissage initial, l'ensemble des données (original) est utilisé entièrement tandis que dans le système d'apprentissage amélioré, un ensemble de données partiel obtenu à partir de l'ensemble original est utilisé,

— Dans un ensemble de données donné, il existe de nombreuses métriques de code, y compris les niveaux de priorité des erreurs (la criticité des erreurs) dans chaque module. Ce qui permet de séparer puis éliminer les erreurs de priorité mineure, pour obtenir de meilleurs résultats expérimentaux,

— Les expériences pour chaque algorithme d'apprentissage automatique sont répétées 200 fois (époques) et les valeurs moyennes des résultats sont prises en compte dans le calcul du taux d'amélioration,

— Pour garantir le caractère aléatoire des données, tous les ensembles de données sont régénérés à partir de l'ensemble de données d'origine en utilisant un algorithme de brassage basé sur un déplacement aléatoire des lignes de l'ensemble de données.

6 Résultats et conclusions obtenus

Les résultats expérimentaux en utilisant les ensembles de données complets sont susceptibles d'avoir un biais dans les estimations concernant les erreurs en raison de la pondération des niveaux de criticité qui n'est pas réaliser de manière égale dans l'ensemble de données. Par contre, les résultats expérimentaux obtenus en utilisant des ensembles de données qui ne contiennent pas de défauts de priorité mineure en utilisant les arbres de décisions (DT), le Perceptron multicouche (MLP) et les fonctions de base radiales (RBF) sont plus satisfaisante. En effet, elles montrent que le système proposé est capable de faire des estimations plus précises en utilisant l'ensemble de données contenant uniquement les erreurs de criticité majeure. En procédant ainsi, les performances du système d'apprentissage sont améliorés.

7 Avis personnel

Les questions suivantes, permettent de structurer notre avis personnel.

1. Quel est votre avis personnel ? (Donnez votre évaluation personnelle de l'approche)
2. Imaginez que vous êtes en entreprise et que vous avez le même problème à traiter, quelle approche proposeriez-vous ?

Pour répondre à la question 1, on rappelle que les chercheurs de l'article choisi, ont supprimé tous les défauts labellisés "mineur" dans le dataset (en grande majorité). En effet, ils considèrent que ces données, induisent les algorithmes à sous-estimé les défauts "majeur". Or ce choix est discutable, puisqu'ils ont choisis de complétement ignorés ces défauts de l'analyse.
Nous pensons, qu'il faudrait pouvoir fournir un mécanisme d'attention aux algorithmes. Ce dernier permettra de quantifier l'attention portée à certains labels dans un dataset. En effet, en tant qu'ingénieur logiciel humain, nous pouvons rechercher des défauts majeurs dans un projet sans pour autant éliminer la détection des défauts mineurs (dont nous restons conscients lors du développement).

Pour répondre à la question 2, il est admis aujourd'hui, qu'un grand nombre d'entreprises possède des outils d'intégration continue de code source en plus d'un système de version, utilisé durant le cycle de vie d'un logiciel.
Parmi ces outils de CI/CD, certains se focalisent sur la qualité du code source et la détection de bugs (labellisé mineur à majeur). Ce qui leur permet de produire des résultats normalisés et analysables. De plus, grâce à l'historique fournit par un logiciel de version, il est possible de savoir quand un bug a été détecté, corrigé, de quelle manière et par quelles personnes.
Nous pouvons, ainsi, imaginer le développement d'un outil de CI/CD (plugin a SonarQube) utilisant des métriques telles qu'un historique des bugs produits dans le passé pour identifier les bugs futurs.
En effet, les prévisions produites par les algorithmes de machine learning pourront être personnalisés pour chaque entreprise et sauvegarder dans un historique. Ce dernier, aura un impact direct sur la précision de la détection de bugs qui devraient être meilleur que celle des approches plus génériques, puisque les exemples produits et à produire, proviennent de la même entreprise et des mêmes équipes de développement. Ce qui est un avantage non négligeable.

8 Conclusion

Dans ce rapport, nous avons lu un article scientifique[1] sur la détection des bugs dans le code source d'un projet logiciel en utilisant des techniques du machine learning.
En se basant sur notre compréhension de ce dernier et le plan défini dans l'énoncé, nous avons synthétisé le contexte et le problème traiter dans cette recherche, les solutions connues de l'article et la solution retenue par l'article, la démarche suivie et les résultats produits. Enfin, nous avons donnés notre avis personnel et suggéré une autre solution pour détection des bugs.

Références bibliographiques

[1] Evren Ceylan, F Onur Kutlubay et Ayse B Bener. « Software defect identification using machine learning techniques ». In : *32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06)*. IEEE. 2006, p. 240-247.